Enhanced spatio-temporal electric load forecasts using less data with active deep learning

Aryandoust, Arsam; Patt, Anthony; Pfenninger, Stefan

DOI
10.1038/s42256-022-00552-x

Publication date
2022

Document Version
Final published version

Published in
Nature Machine Intelligence

nature machine intelligence

Article

https://doi.org/10.1038/s42256-022-00552-x

Enhanced spatio-temporal electric load forecasts using less data with active deep learning

Received: 8 December 2020

Accepted: 28 September 2022

Published online: 15 November 2022

Check for updates

Arsam Aryandoust [1] ✉, Anthony Patt[1] & Stefan Pfenninger[1,2]

An effective way to mitigate climate change is to electrify most of our energy demand and supply the necessary electricity from renewable wind and solar power plants. Spatio-temporal predictions of electric load become increasingly important for planning this transition, while deep learning prediction models provide increasingly accurate predictions for it. The data that are used for training deep learning models, however, are usually collected at random using a passive learning approach. This naturally results in a large demand for data and associated costs for sensors such as smart meters, posing a large barrier for electric utilities when decarbonizing their grids. Here we investigate whether electric utilities can use active learning to collect a more informative subset of data by leveraging additional computation for better distributing smart meters. We predict ground-truth electric load profiles for single buildings using only remotely sensed data from aerial imagery of these buildings and meteorological conditions in the area of these buildings at different times. We find that active learning can enable 26–81% more accurate predictions using 29–46% less data at the price of 4–11 times more computation compared with passive learning.

An effective way to mitigate climate change is to electrify our energy sectors and supply their electricity from renewable wind and solar, which are highly fluctuating and uncertain sources of energy[1–3]. Planning and operating electricity grids under these uncertainties increasingly requires fine-grained and accurate predictions of electric load across very short to long time windows[4,5]. Among the different types of electric load forecasts that are performed[6,7], spatio-temporal predictions have gained increasing importance[8–10]. They predict load for times and places for which we do not have detailed information about electric load profiles in our grids, and operate these as black boxes[11].

While remotely sensed data such as meteorological conditions or satellite imagery are increasingly easy to access for making spatio-temporal predictions[12,13], ground-truth electric load data remain difficult and expensive to collect[14]. One reason for this is that electric utilities can be limited in the number of physical sensors such as smart meters that they can place to collect load data owing to technical, financial and social barriers[15,16]. Another reason is that utilities can further be limited in the amount of data they can query from each meter in real time by constraints such as data communication band widths and privacy concerns of consumers, which is known as the velocity constraint of data[17,18].

Figure 1 shows the state of global smart meter adoption. In regions of the world with medium to high adoption of smart meters, we want to know when to query data from which meter so as to best utilize our measured data for making accurate predictions of load without exceeding our data velocity constraints. In regions of the world with a yet low adoption of smart meters, we further want to know where to install new meters first so as to make the best possible predictions of load for the parts of our grids that remain unmeasured.

Artificial intelligence (AI) and machine learning (ML) are increasingly used to tackle such climate change-related prediction problems[19,20]. For example, AI is used in 90% of recently proposed load

[1]Climate Policy Lab, Department of Environmental Systems Science, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland. [2]Faculty of Technology, Policy and Management (TPM), Delft University of Technology, Delft, the Netherlands. ✉e-mail: arsama@ethz.ch

https://doi.org/10.1038/s42256-022-00552-x

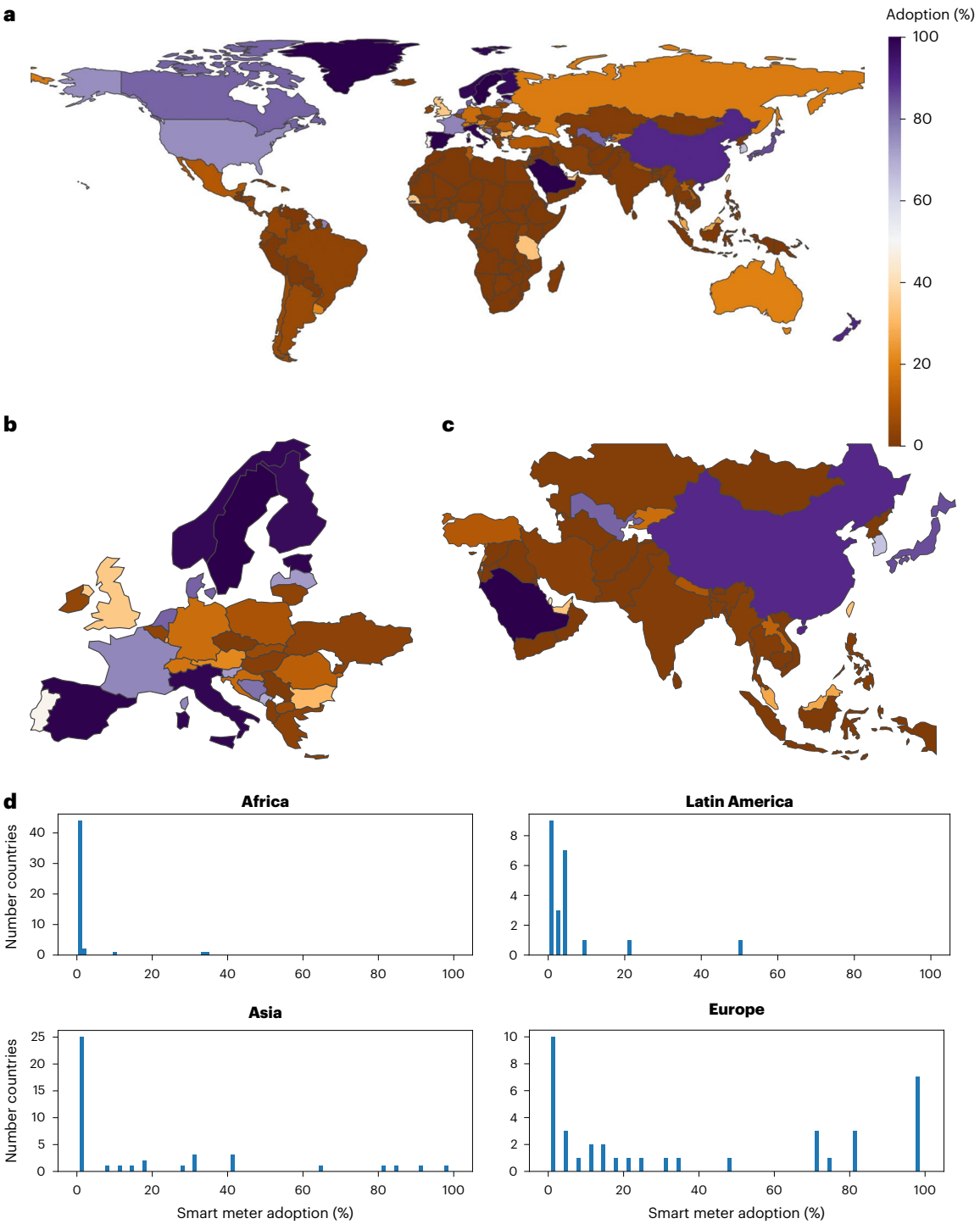

Fig. 1 | The current state of global smart meter adoption. a–c, Smart meter adoption worldwide (**a**) and in Europe (**b**) and Asia (**c**). A diverging colour spectrum is chosen to better contrast the difference between neighbours. **d**, The distribution of adoption by country for Africa, Latin America, Asia and Europe.

forecasting algorithms, where deep learning (DL) models make up the largest share with 28% (ref. [7]). The default method of choice for training these DL models is passive learning. In passive learning, the data used for training a prediction model are collected at random from a large pool of candidate data points, which naturally results in a large demand for data and sensors in remote sensing applications. In contrast, an increasingly emerging approach, known as active learning, leverages additional computation to assess the information content of data points before collecting these, such that we can collect the most useful data only and reduce our overall demand for data and sensors. Supplementary Note 1 provides more detail on how AI, ML and DL encompass each other and how active learning has emerged to solve problems associated with passive learning.

Although the advantage of active deep learning (ADL) over passive deep learning (PDL) is well studied across many theoretical use cases and domains, its application for solving important real-world problems such as collecting data for spatio-temporal predictions of electric load remains poorly explored. As one of the first applications of

ADL in a related problem domain, Kuo et al. reduced the computational complexity of predicting electricity prices by sampling a smaller, more informative subset of training data from a large pool of candidate data points using Gaussian processes[21]. Wang et al. increased their model accuracy for time series predictions of electric load compared with existing DL models by using a selector–predictor framework in which the selector samples a subset of similar and correlating load segments from the past to be used as training data by a distinct and continuously updated predictor that consists of an ensemble of DL models[22]. Zhang et al. reduced their dataset bias and thus also their overall generalization errors for time series predictions of electric load by sampling a training subset from a candidate data pool of past consumption data that is more representative of the entire data population based on reducing an expected error metric[23].

Here, we investigate (1) how a dataset of the same size picked using ADL impacts the informativeness of training data and generalization performance compared with when using PDL, (2) how much data and how many sensors we can save when using ADL while making as accurate predictions as when using PDL, (3) how much additional computation is required when using ADL as compared with when using PDL and (4) how the sequence of training data picked with ADL impacts the generalization performance. In the following, we provide an overview of our prediction task and ADL method, before presenting and discussing our results.

Prediction task and ADL method

Predictions of electric load at different spatio-temporal resolutions are useful for planning and operating active distribution grids that will not only consume but also produce electricity through the integration of higher shares of wind and solar power throughout our energy transition[24,25]. Here, we exemplarily perform spatio-temporal predictions of load at the scale of entire buildings. This allows utilities to study how load profiles change when electrifying additional energy demands such as heating and mobility, how much of a building's electricity demand can be covered from roof-top and nearby wind and solar power, and how large resulting residual loads and bi-directional power flows in active distribution grids can be.

Given the aerial image of a building, the meteorological conditions in the region of that building and a time stamp as our features, we want to predict the electric load profile of a building for the next 24 h in 15 min steps as our labels. Our features (inputs) are all remotely sensed and assumed to be available for every building and point in time at no cost. For every new load profile (output or label) that we collect, we experience some cost and are constrained in the total number of profiles that we can collect by some budget n_{budget}. We start with a prediction model that has learnt this relationship for a few buildings and times. Our goal is to collect further ground-truth data, that is, the electric load profiles at different times and buildings, so as to make the best possible predictions for buildings and times for which we do not have load profiles available, without exceeding n_{budget}. Supplementary Note 2 explains the implications of choosing our data in this constellation in more detail. We run experiments on two different datasets: one containing load profiles from 100, and one from 400 residential, commercial and industrial buildings in Switzerland with diverse sizes, shapes, occupancy and consumption. For each experiment, we randomly select load profiles from 800 (for the dataset with 100 buildings) and 200 (for the dataset with 400 buildings) time stamps in 2014 to create a candidate data pool that is of about the same size for each dataset. Figure 2 shows the modular DL model which contains a novel architecture that we propose for solving this task. We call this a spatio-temporal embedding network.

In each iteration of the ADL algorithm that we apply, we query a batch of candidate data points. First, we encode the features of candidate data points into an embedded vector space using our spatio-temporal embedding network (Fig. 2) that is trained on initially available random data points. We then cluster candidate data points based on their vector distances from each other in this encoded space, with the number of clusters being equal to our query batch size. Next, we calculate the distance of the vector of each encoded data point to its cluster centre and query one data point per cluster based on these distances. We test our ADL method for randomized, minimized, maximized and averaged distances of embedded data points to their cluster centres in every queried data batch. We refer to these as our ADL variants. A number of alternative active learning methods exist that we can apply to our data selection task. Supplementary Note 3 describes these in further detail and explains the advantage and potential disadvantage of using the active learning method we propose over existing ones.

Figure 3 visualizes the difference between data queries with each of our ADL variants. In a first variant, we randomly select data points from each embedded cluster of candidates (Fig. 3a). In a second variant, we query candidate data points whose embedded feature vectors are furthest away from their cluster centres (Fig. 3b). We expect to be more uncertain about these points, as they are more likely to be true members of another cluster. We likely explore the data that are close to our decision boundaries, if not outliers, and expect a larger surprise or learning experience from querying labels for these data points. In a third variant, we query labels of data points that are close to their cluster centres, which we expect to be more representative of their clusters and respectively our entire data population (Fig. 3c). In a fourth variant, we query data points that have the largest distance to the average of distances to cluster centres among all points of the same cluster, which results in a combination of queries changing between uncertain and representative data points (Fig. 3d). Each of these ADL variants tries to select a subset of data points from the candidate data pool with a different policy that is more informative compared with when selecting these uniformly at random using PDL. The distance of candidate data points to their cluster centres in an embedded vector space is a new metric of informativeness that we propose. We call it the embedding uncertainty.

We evaluate the performance of our ADL and PDL algorithms for spatial, temporal and spatio-temporal predictions compared with a random forest (RF) benchmark as a common reference. In this context, temporal predictions mean that we predict load profiles for buildings in which a smart meter is placed, but for a time period into the past or future for which we do not have measured data available. This allows us to compare the prediction performance against a distribution shift of our data in time only. Spatial predictions mean that we predict load profiles for buildings in which a smart meter is not placed but for a time period in which we have load profiles available for other buildings. This allows us to compare the prediction performance against a distribution shift of our data in space only. Spatio-temporal predictions are the most difficult problem of predicting load profiles for times and buildings for which we do not have any load profiles available at all. This allows us to compare the prediction performance against a distribution shift of our data in both time and space. We refer to these as the different prediction types that we evaluate.

For each prediction type that we evaluate, we further distinguish between the type of features that we can encode for querying candidate data points. We distinguish between features that are variant in time $\mathbf{x}_t$ (time stamp), space $\mathbf{x}_s$ (building image) and both time and space $\mathbf{x}_{st}$ (meteorological data), as well as the entire feature vector $\mathbf{x}_{t,s}$ which is concatenated from these three vectors. As the predicted output of our network $\hat{\mathbf{y}}_{t,s}$ represents a vector, that is, the electric consumption of a building for the next 24 h in 15 min steps (96 values), we can also use this vector as an embedding of our entire feature vector $\mathbf{x}_{t,s}$. In a further test, we hypothetically use our true labels $\mathbf{y}_{t,s}$ for querying candidate data points in order to see how our proposed metric and ADL variants perform with knowledge about the true distances or similarities of labels that we try to otherwise infer from our features only. We refer to these as our ADL variables.

https://doi.org/10.1038/s42256-022-00552-x

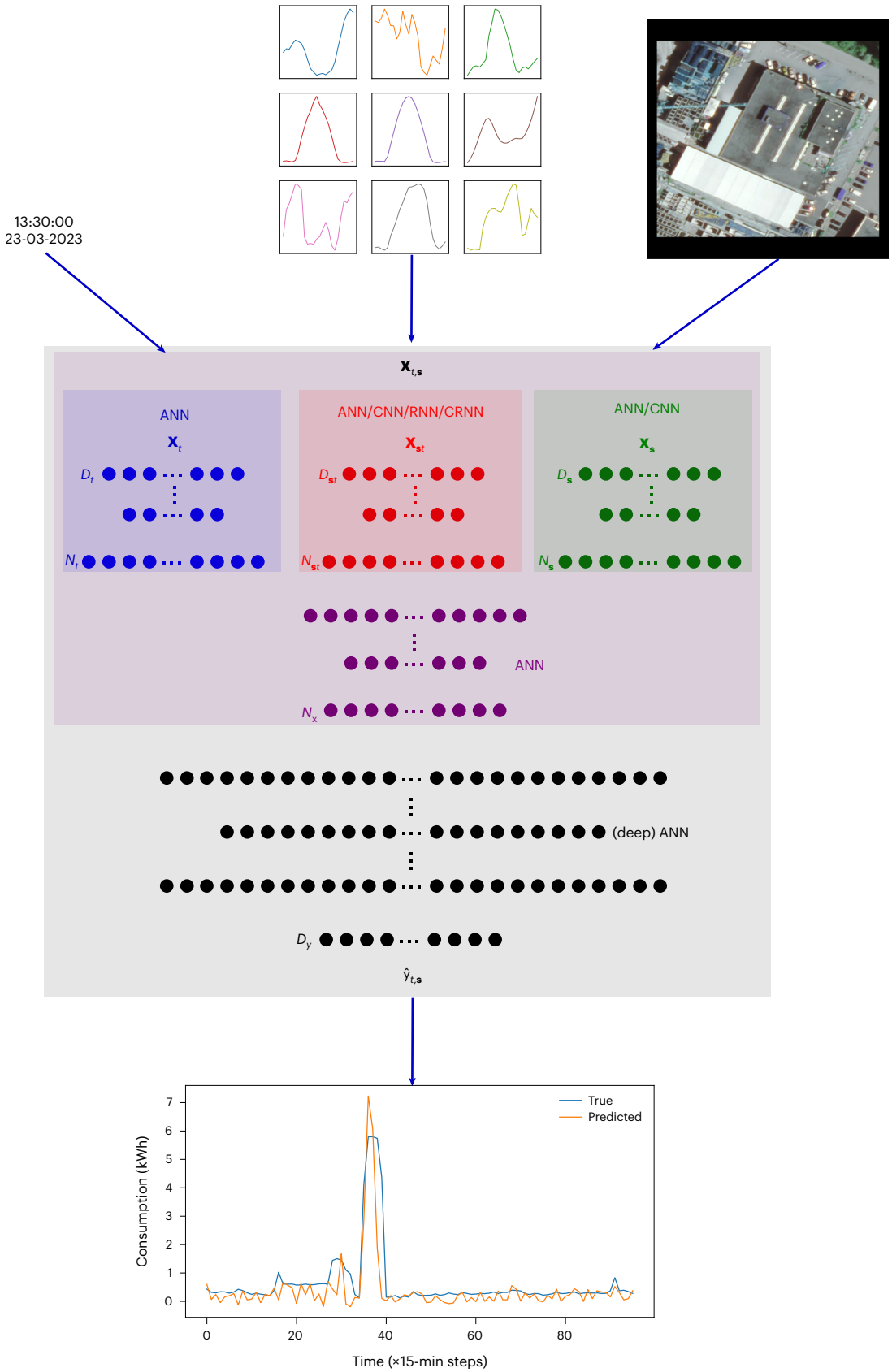

Fig. 2 | Overview of the spatio-temporal embedding network architecture. The structure is based on densely connected neural network (ANN), convolutional neural network (CNN), recurrent neural network (RNN) and convolutional and recurrent neural network (CRNN) architectures. The inputs or features in this prediction task consist of a time stamp, an aerial image of a building and nine meteorological conditions from the region of that building at a given time and place. The output or label is the electric consumption of that building at that given time for the next 24 h in 15-min steps (96 values).

https://doi.org/10.1038/s42256-022-00552-x

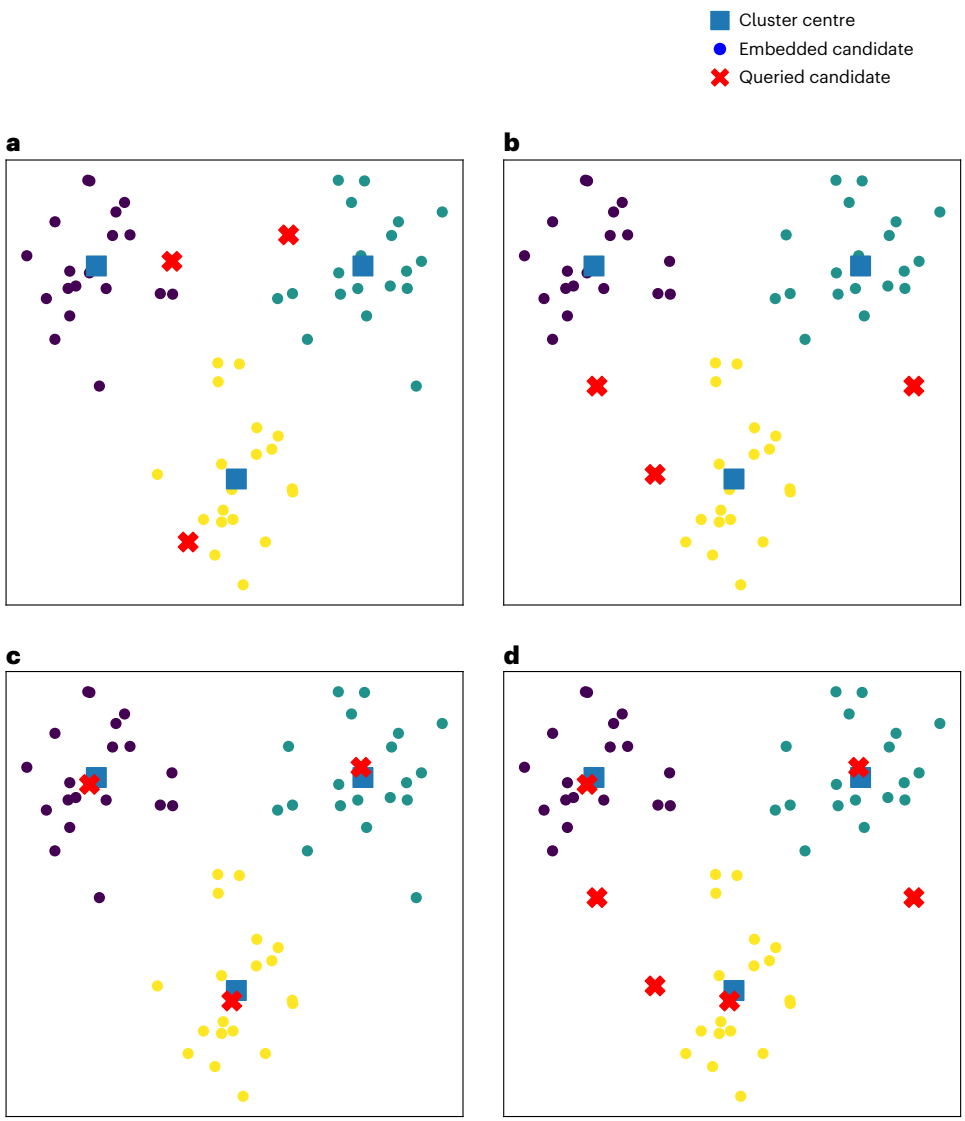

Fig. 3 | Overview of ADL variants. a–**d**, The plot shows exemplary data points embedded into a two-dimensional vector space for visualization. The entries of the embedded data vectors are represented on each axes. Dots of the same colour represent embedded data points that are classified into the same cluster. Variants when randomizing (**a**), maximizing (**b**), minimizing (**c**) and averaging (**d**) the embedding uncertainty of a queried data batch in one iteration of our algorithm (in **d**, only one point is queried per cluster).

In all experiments, we remove queried data from the candidate data pool at a rate of $\delta \in [0, 1]$, and test this for the two extreme cases of $\delta = 0$ and $\delta = 1$. Removing all queried data from the candidate pool ($\delta = 1$) forces us to exploit our entire data budget and allows us to test how the same number of new data points picked with ADL impacts the prediction accuracy compared with when using PDL, and answer our first research question. Keeping all queried data in the candidate pool ($\delta = 0$) allows our ADL and PDL algorithms to query a multiset from our candidate pool and achieve data and sensor savings. This lets us test how much data and how many sensors we can save with ADL while making as accurate a prediction as when using PDL, and answer our second research question. We further subsample the candidate data pool, reduce the number of clusters in the candidate data pool and query candidates by their spatial coordinate embedding only to evaluate how our computational complexity can be decreased, to better answer our third research question. Our fourth research question is answered by randomizing the sequence of our ADL sample set and comparing the results against our original sampling sequences.

Results

When removing queried data from our candidate data pool such that our entire data budget is used ($\delta = 1$), our prediction accuracy increases up to 36–74%, using 4–8 times more computation, with ADL compared with when using PDL. When keeping queried data points in the candidate data pool instead ($\delta = 0$), our demand for data reduces by up to 29–46% while achieving up to 26–81% higher prediction accuracy using 4–11 times more computation. Our demand for sensors can be reduced by up to 24% with an increase in prediction accuracy of up to 35%.

Table 1 presents the numerical results for each prediction type, ADL variable and ADL variant on a dataset with 400 buildings. Supplementary Table 1 further contains the numerical results for the experiments on a dataset with 100 buildings. We can observe a correlation between the leveraged computation and the increased prediction accuracy and data savings. Figure 4 shows the training and validation losses for the main experiments that we run to answer research questions 1 and 2. Figure 5 contains only the validation losses for the additional experiments that we run to answer research questions 3 and 4. In the following, we describe each experiment and their key observations in detail.

https://doi.org/10.1038/s42256-022-00552-x

Table 1 | Numerical results for each prediction type, ADL variable and ADL variant with experiments on the dataset with 400 buildings

			Removing queried data (δ=1)				Keeping queried data (δ=0)			
Prediction type	Variable	Variant	Compute	Data	Sensors	Accuracy	Compute	Data	Sensors	Accuracy
Spatial	$\mathbf{x}_{st}$	rnd $d_{c,st}$	3.2×	100%	100%	43%	3.3×	70%	100%	33%
		min $d_{c,st}$	3.3×	100%	100%	46%	3.4×	57%	99%	24%
		max $d_{c,st}$	3.3×	100%	95%	22%	3.4×	27%	55%	0%
		avg $d_{c,st}$	3.4×	100%	95%	26%	3.4×	28%	54%	20%
	$\mathbf{x}_{t,s}$	rnd $d_{c,(t,s)}$	3.9×	100%	100%	75%	4.1×	75%	100%	68%
		min $d_{c,(t,s)}$	4.1×	100%	100%	71%	4.2×	75%	100%	56%
		max $d_{c,(t,s)}$	4.2×	100%	100%	73%	4.2×	54%	91%	59%
		avg $d_{c,(t,s)}$	4.3×	100%	100%	79%	4.4×	62%	100%	64%
	$\hat{\mathbf{y}}_{t,s}$	rnd $d_{c,(t,s)}$	4.4×	100%	100%	86%	5.0×	48%	97%	80%
		min $d_{c,(t,s)}$	4.6×	100%	99%	83%	5.0×	50%	95%	80%
		max $d_{c,(t,s)}$	4.5×	100%	100%	70%	4.9×	34%	76%	71%
		avg $d_{c,(t,s)}$	4.8×	100%	98%	75%	4.7×	41%	92%	79%
	$\mathbf{y}_{t,s}$	rnd $d_{c,(t,s)}$	4.3×	100%	98%	87%	4.6×	42%	92%	63%
		min $d_{c,(t,s)}$	4.1×	100%	96%	86%	4.8×	28%	64%	47%
		max $d_{c,(t,s)}$	4.3×	100%	95%	85%	4.8×	23%	55%	10%
		avg $d_{c,(t,s)}$	4.2×	100%	94%	85%	4.7×	27%	66%	37%
PDL benchmark			1.0×	100%	100%	50%	1.0×	80%	100%	36%
Temporal	$\mathbf{x}_{st}$	rnd $d_{c,st}$	3.2×	100%	0%	12%	4.7×	76%	0%	15%
		min $d_{c,st}$	3.3×	100%	0%	6%	4.8×	53%	0%	0%
		max $d_{c,st}$	3.2×	100%	0%	0%	4.7×	33%	0%	0%
		avg $d_{c,st}$	3.3×	100%	0%	0%	5.1×	38%	0%	0%
	$\mathbf{x}_{t,s}$	rnd $d_{c,(t,s)}$	3.4×	100%	0%	29%	6.8×	75%	0%	25%
		min $d_{c,(t,s)}$	4.0×	100%	0%	39%	7.0×	70%	0%	31%
		max $d_{c,(t,s)}$	3.9×	100%	0%	32%	6.9×	55%	0%	16%
		avg $d_{c,(t,s)}$	3.8×	100%	0%	34%	6.8×	63%	0%	28%
	$\hat{\mathbf{y}}_{t,s}$	rnd $d_{c,(t,s)}$	4.0×	100%	0%	81%	6.8×	51%	0%	49%
		min $d_{c,(t,s)}$	4.0×	100%	0%	81%	6.8×	52%	0%	46%
		max $d_{c,(t,s)}$	4.2×	100%	0%	83%	7.3×	40%	0%	38%
		avg $d_{c,(t,s)}$	4.2×	100%	0%	83%	7.0×	44%	0%	44%
	$\mathbf{y}_{t,s}$	rnd $d_{c,(t,s)}$	4.1×	100%	0%	77%	7.4×	45%	0%	22%
		min $d_{c,(t,s)}$	3.9×	100%	0%	73%	7.2×	33%	0%	3%
		max $d_{c,(t,s)}$	3.9×	100%	0%	78%	7.1×	26%	0%	0%
		avg $d_{c,(t,s)}$	4.3×	100%	0%	73%	7.3×	31%	0%	0%
PDL benchmark			1.0×	100%	0%	9%	1.0×	80%	0%	9%
Spatio-temporal	$\mathbf{x}_{st}$	rnd $d_{c,st}$	5.5×	100%	100%	34%	6.7×	70%	100%	42%
		min $d_{c,st}$	5.5×	100%	100%	36%	6.9×	63%	100%	43%
		max $d_{c,st}$	5.7×	100%	97%	16%	6.9×	35%	75%	8%
		avg $d_{c,st}$	5.6×	100%	98%	21%	7.0×	31%	62%	4%
	$\mathbf{x}_{t,s}$	rnd $d_{c,(t,s)}$	7.1×	100%	100%	63%	9.0×	77%	100%	57%

https://doi.org/10.1038/s42256-022-00552-x

Table 1 (continued) | Numerical results for each prediction type, ADL variable and ADL variant with experiments on the dataset with 400 buildings

			Removing queried data (δ=1)				Keeping queried data (δ=0)			
Prediction type	**Variable**	**Variant**	**Compute**	**Data**	**Sensors**	**Accuracy**	**Compute**	**Data**	**Sensors**	**Accuracy**
		min $d_{c,(t,s)}$	7.2×	100%	100%	49%	8.7×	74%	100%	60%
		max $d_{c,(t,s)}$	7.0×	100%	100%	67%	9.4×	59%	100%	49%
		avg $d_{c,(t,s)}$	7.1×	100%	100%	71%	9.3×	65%	100%	50%
	$\hat{\mathbf{y}}_{t,s}$	rnd $d_{c,(t,s)}$	8.5×	100%	100%	88%	10.7×	46%	100%	73%
		min $d_{c,(t,s)}$	8.5×	100%	100%	87%	10.6×	49%	100%	71%
		max $d_{c,(t,s)}$	8.3×	100%	100%	88%	10.5×	40%	97%	68%
		avg $d_{c,(t,s)}$	8.2×	100%	100%	88%	10.2×	40%	100%	69%
	$\mathbf{y}_{t,s}$	rnd $d_{c,(t,s)}$	7.8×	100%	100%	85%	10.3×	36%	98%	59%
		min $d_{c,(t,s)}$	8.1×	100%	96%	79%	10.3×	27%	73%	10%
		max $d_{c,(t,s)}$	7.6×	100%	91%	84%	10.2×	23%	49%	0%
		avg $d_{c,(t,s)}$	7.9×	100%	95%	82%	10.3×	26%	66%	0%
PDL benchmark			1.0×	100%	100%	40%	1.0×	80%	100%	43%

The 'compute' columns present the factor for additional computation performed. The 'data' columns present what percentage of the data budget was used. The 'sensors' columns state what percentage of sensors was used from the new sensors initially available in the candidate data pool. The 'accuracy' is calculated as $1 - \min\left(1, \frac{\text{PDL loss}}{\text{RF loss}}\right)$ and $1 - \min\left(1, \frac{\text{ADL loss}}{\text{RF loss}}\right)$. The variants represent data queries with random (rnd), minimized (min), maximized (max) and average (avg) embedding uncertainty.

Training and validation losses when removing all queried data ($\delta = 1$)

We validate predictions on unqueried data points during training and remove all queried data points from the candidate data pool (Fig. 4a,b). Training (Fig. 4a) and validation (Fig. 4b) losses are visualized for all ADL variants of an exemplar ADL variable compared with PDL (dashed blue line). With each newly queried batch, we can observe that training losses leap up (Fig. 4a) while validation losses drop (Fig. 4b). We can further observe that a correlation exists between the magnitude of leaps in training loss and drops in validation loss, which are generally larger for ADL than for PDL. Large leaps in training loss indicate that we query data points that are diverse and 'hard to fit to'. We implicitly regularize our prediction model by not overfitting to 'easy to learn' patterns in the queried data batch, which lets us generalize better on our unqueried data population.

Training and validation losses when keeping all queried data ($\delta = 0$)

Next, we validate predictions on unqueried data points during training and keep queried data points in the candidate data pool (Fig. 4c,d). Training (Fig. 4c) and validation (Fig. 4d) losses are visualized for all ADL variants of an exemplar ADL variable compared with PDL (dashed blue line). Unlike in the case of removing all queried candidates, training losses converge to a value that is higher than for PDL (Fig. 4c). The gap between ADL and PDL validation losses (Fig. 4d) is smaller compared with when removing queried data from the candidate data pool. This indicates a stronger weight regularization as compared with when removing queried data. We achieve data savings at simultaneously higher prediction accuracy.

Validation losses against queried and unqueried candidates

We validate the prediction accuracy for queried and unqueried data (Fig. 4e–h) to see how introducing a sampling bias through values of $0 < \delta \leq 1$ impacts the prediction accuracy on our entire data population. With $\delta = 1$ (Fig. 4e,f), we create a tendency to forget previously learnt information upon learning new information, which results in better predictions on remaining candidates and less accurate predictions on already queried candidates. This is seen from the lower testing losses compared with $\delta = 0$ and an increase of validation losses towards the final iterations. With $\delta = 0$ (Fig. 4g,h), our model remains resilient against such tendencies. We observe higher testing losses on yet unqueried candidates. We conclude that values of $\delta \to 1$ implicitly cause 'catastrophic inference' or 'forgetting' upon learning new information, which increases the prediction accuracy on our unqueried data population, while values of $\delta \to 0$ allow 'incremental learning' with better predictions on the total data population.

Computational complexity

We reduce the candidate pool size, number of clusters or number of embedded coordinates to achieve faster computation (Fig. 5a–f). Subsampling the candidate pool (Fig. 5a,b) and reducing the number of clusters in the pool (Fig. 5c,d) by up to 75% both reduce our computational complexity by a factor of 2–3 times with only a slight or no decrease in prediction accuracy and data savings, but with a higher 'regret'. When querying candidates by their embedded spatial coordinates (Fig. 5e,f), we observe only 10–90% additional computation compared with PDL. When removing queried data from the candidate pool (Fig. 5e), we can achieve about a 20% increase in prediction accuracy using the same amount of data, with about half the number of sensors. When keeping queried data in the candidate pool (Fig. 5f), we query too few data points and sensors to make better predictions than PDL.

Importance of ADL query sequence

We compare validation losses during ADL with the case in which we randomize the sequence of queries with the same (multi)set of data during training (Fig. 5g,h). With $\delta = 1$ (Fig. 5g), the original ADL sequences converge faster to their final generalization errors than when querying the same data in a random sequence. With $\delta = 0$ (Fig. 5h), our prediction model mostly remains invariant towards the query sequence of candidates. We therefore observe that values of $\delta \to 1$ achieve a lower online learning 'regret' than values of $\delta \to 0$.

https://doi.org/10.1038/s42256-022-00552-x

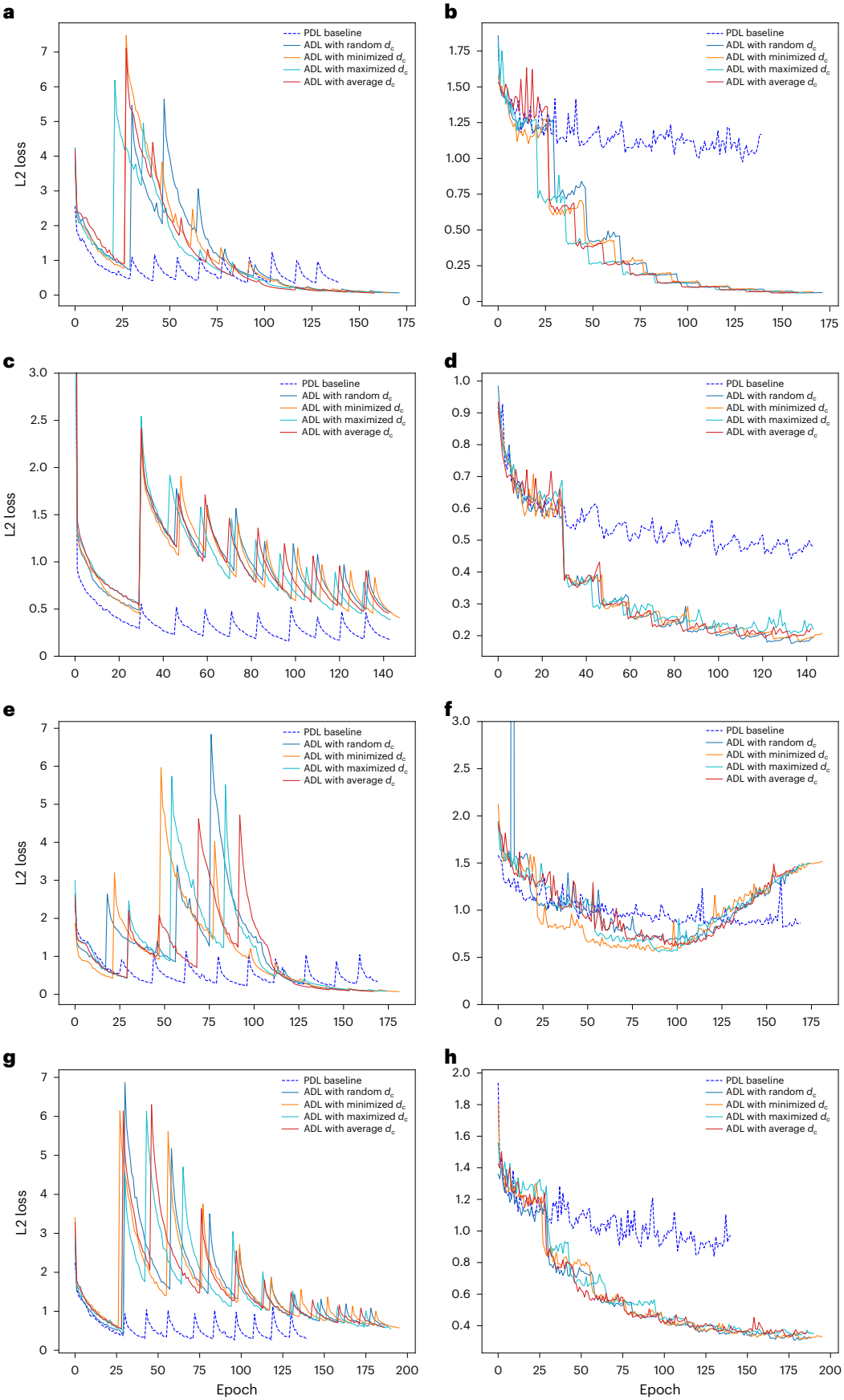

Fig. 4 | Exemplar results for spatio-temporal predictions and ADL variable $\hat{\mathbf{y}}_{t,s}$. a–**h**, Training losses (**a**,**c**,**e**,**g**) and validation losses (**b**,**d**,**f**,**h**) against remaining candidates with $\delta = 1$ (**a**,**b**) or $\delta = 0$ (**c**,**d**) and against entire data population with $\delta = 1$ (**e**,**f**) or $\delta = 0$ (**g**,**h**).

https://doi.org/10.1038/s42256-022-00552-x

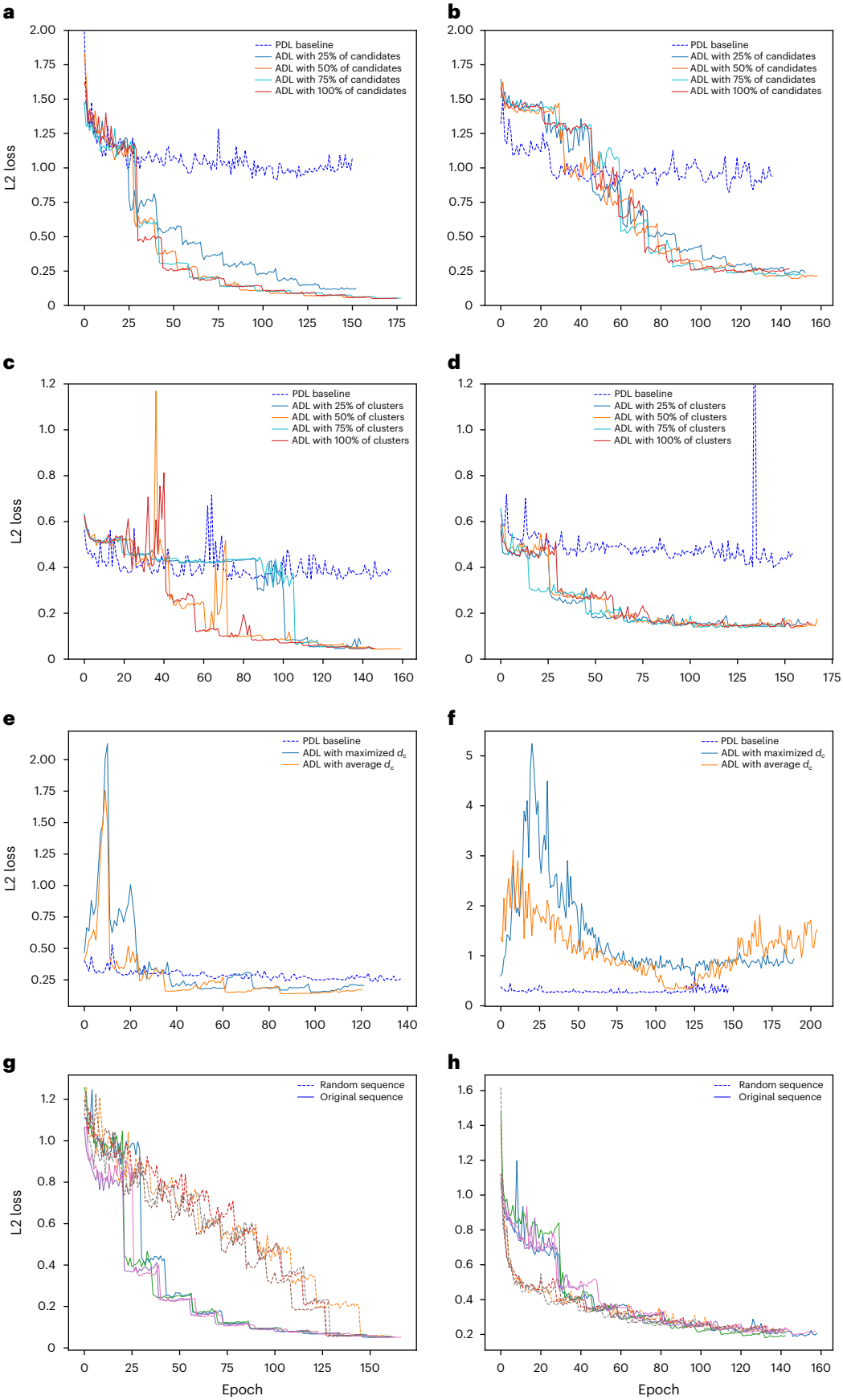

Fig. 5 | Exemplar results for reducing computational time and testing query sequence. a–**h**, Reductions achieved by subsampling candidate data (**a**,**b**), reducing the number of clusters (**c**,**d**), querying candidates on the basis of spatial coordinate embedding (**e**,**f**), and evaluating the sequence importance of queries (**g**,**h**), showing only validation losses with $\delta = 1$ (**a**,**c**,**e**,**g**) or $\delta = 0$ (**b**,**d**,**f**,**h**).

Discussion

We investigate (1) how a dataset of the same size picked with ADL impacts the informativeness of training data and generalization performance compared with when using PDL, (2) how much data and how many sensors we can save when using ADL while making as accurate predictions as when using PDL, (3) how much additional computation we require when using ADL as compared with when using PDL and (4) whether the sequence of training data picked with ADL has an impact on the generalization performance, for predicting electric load profiles of single buildings in time and space. Surprisingly, we find that we can achieve an even higher prediction accuracy with significantly less data when leveraging additional computation for selecting a more informative data subset with ADL.

We increase the prediction accuracy by up to 36–74% when using our entire data budget (question 1). Our demand for data reduces by up to 29–46% while achieving an up to 26–81% higher prediction accuracy, and our demand for smart meters can be reduced by up to 24% with an up to 35% higher prediction accuracy (question 2). We achieve these improvements at the cost of about 4–11 times more computation for assessing the informativeness of each candidate data point. We can reduce this complexity by a factor of 2–3 by subsampling our candidate pool or reducing the number of clusters, at the price of a higher online learning regret. When querying candidates by the embedding of spatial coordinates, we still achieve about 20% accuracy improvements over PDL, but with only 10% more computation (question 3). We find that the sequence in which data are selected for ADL is further meaningful. Training models with data selected in a sequence picked by our ADL method creates a lower online learning regret compared with when randomizing the sequence of the same set of data during training (question 4).

Our findings can have important implications for the clean energy transition and for mitigating climate change. We show that electric utilities around the world can use ADL instead of PDL to make more accurate predictions of load by distributing new smart meters and streaming their data more effectively. For the general reliability and applicability of our findings, our proposed ADL method must be tested on a larger variety of datasets and prediction tasks and be compared against alternative ADL methods. Further research can explore how contrastive learning[26] and domain adaptation[27–29] can further reduce data and sensor demand while increasing prediction accuracy before applying ADL, and how ADL can be used for spatio-temporal predictions with graph neural networks[30] to better consider distribution shifts in space–time.

Methods

The spatio-temporal prediction problem

Given a map of the Earth, we want to predict some value of interest $\mathbf{y}_{t,\mathbf{s}} \in \mathbb{R}^{D_y}$ of dimension $D_y \in \mathbb{Z}^+$ in time $t \in \mathbb{N}$ and space $\mathbf{s} \in \mathbb{R}^2$ such that $\mathbf{s} = (lat, long)$, with $lat \in [-90, 90]$ and $long \in [-180, 180]$. The ranges of the variables *lat* and *long* refer to the possible values of geographic latitudinal and longitudinal coordinates. Hereby, the starting point in time and the accuracy in both time and space are application dependent and can be chosen arbitrarily. The set of all $\mathbf{y}_{t,\mathbf{s}}$, hereafter called labels, is referred to as $\mathcal{Y}$. Each label is hence a vector

$$\mathbf{y}_{t,\mathbf{s}} = \begin{pmatrix} y_{t,\mathbf{s},1} \\ \vdots \\ y_{t,\mathbf{s},D_y} \end{pmatrix}.$$

Given the features $\mathbf{x}_{t,\mathbf{s}} \in \mathbb{R}^{D_x}$ of dimension $D_x \in \mathbb{Z}^+$ for each label, we want to predict labels for particular points of interest in time and space. We refer to the set of all features as $\mathcal{X}$. Each label hence has a corresponding feature vector

$$\mathbf{x}_{t,\mathbf{s}} = \begin{pmatrix} x_{t,\mathbf{s},1} \\ \vdots \\ x_{t,\mathbf{s},D_x} \end{pmatrix}.$$

We can further classify the single entries of $\mathbf{x}_{t,\mathbf{s}}$ as space, time and space–time variant features. Features that are constant in time t but variant in space $\mathbf{s}$ are referred to as space-variant features $\mathbf{x}_\mathbf{s} \in \mathbb{R}^{D_s}$ of dimension $D_s \in \mathbb{N}$ such that $D_s \leq D_x$:

$$\mathbf{x}_\mathbf{s} = \begin{pmatrix} x_{t,\mathbf{s},1} \\ \vdots \\ x_{t,\mathbf{s},D_s} \end{pmatrix} = \begin{pmatrix} x_{t+\hat{t},\mathbf{s},1} \\ \vdots \\ x_{t+\hat{t},\mathbf{s},D_s} \end{pmatrix} \forall \hat{t} \in \mathbb{Z} : 0 \leq t + \hat{t}.$$

Features that are constant in space $\mathbf{s}$ but variant in time t are referred to as time-variant features $\mathbf{x}_t \in \mathbb{R}^{D_t}$ of dimension $D_t \in \mathbb{N}$ such that $D_t \leq D_x$:

$$\mathbf{x}_t = \begin{pmatrix} x_{t,\mathbf{s},D_s+1} \\ \vdots \\ x_{t,\mathbf{s},D_s+D_t} \end{pmatrix} = \begin{pmatrix} x_{t,\mathbf{s}+\hat{\mathbf{s}},D_s+1} \\ \vdots \\ x_{t,\mathbf{s}+\hat{\mathbf{s}},D_s+D_t} \end{pmatrix}$$

$$\forall \hat{\mathbf{s}} \in \mathbb{R}^2 : lat + \hat{lat} \in [-90, 90] \wedge long + \hat{long} \in [-180, 180]$$

Features that are variant in both time t and space $\mathbf{s}$ are referred to as space–time variant features $\mathbf{x}_{st} \in \mathbb{R}^{D_{st}}$ of dimension $D_{st} \in \mathbb{N}$ such that $D_{st} \leq D_x$:

$$\mathbf{x}_{st} = \begin{pmatrix} x_{t,\mathbf{s},D_s+D_t+1} \\ \vdots \\ x_{t,\mathbf{s},D_s+D_t+D_{st}} \end{pmatrix} \neq \begin{pmatrix} x_{t+\hat{t},\mathbf{s}+\hat{\mathbf{s}},D_s+D_t+1} \\ \vdots \\ x_{t+\hat{t},\mathbf{s}+\hat{\mathbf{s}},D_s+D_t+D_{st}} \end{pmatrix}$$

$$\forall \hat{t} \in \mathbb{Z}, \hat{\mathbf{s}} \in \mathbb{R}^2 : 0 \leq t + \hat{t} \wedge lat + \hat{lat} \in [-90, 90] \wedge long + \hat{long} \in [-180, 180]$$

For the above to be valid, it further has to hold that

$$D_s + D_t + D_{st} = D_x$$

and

$$\mathbf{x}_{t,\mathbf{s}} = \text{concat}(\mathbf{x}_t, \mathbf{x}_\mathbf{s}, \mathbf{x}_{st}).$$

We can further distinguish the type of predictions that we make in the same fashion. Let $\mathcal{Y}^{\text{avail}} \subset \mathcal{Y}$ be a subset of all labels that are available to us, that is, the ground-truth values that we have already measured. Then, for a given point $\mathbf{s}$ in space, if we use any knowledge about $\mathbf{y}_{t_1,\mathbf{s}}$ such that t_1 is represented in elements of $\mathcal{Y}^{\text{avail}}$ to make predictions about any $\mathbf{y}_{t_2,\mathbf{s}}$ such that t_2 is not represented by elements of $\mathcal{Y}^{\text{avail}}$, we call this a temporal prediction. Similarly, for a given point t in time, if we use any knowledge about $\mathbf{y}_{t,\mathbf{s}_1}$ such that $\mathbf{s}_1$ is represented by elements of $\mathcal{Y}^{\text{avail}}$ to make predictions about any $\mathbf{y}_{t,\mathbf{s}_2}$ such that $\mathbf{s}_2$ is not represented by elements of $\mathcal{Y}^{\text{avail}}$, we call this a spatial prediction. If we make predictions about any $\mathbf{y}_{t,\mathbf{s}}$ such that $(t, \mathbf{s})$ is not represented by elements of $\mathcal{Y}^{\text{avail}}$, we call this a spatio-temporal prediction.

Furthermore, let $\mathcal{X}^{\text{avail}}$ be the set of features that are complement to each element of $\mathcal{Y}^{\text{avail}}$. Then, we let $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$ be the dataset that consists of all feature–label pairs that exist and let $\mathcal{D}^{\text{avail}} = (\mathcal{X}^{\text{avail}}, \mathcal{Y}^{\text{avail}}) \subset \mathcal{D}$ be a subset that is available to us.

Spatio-temporal embedding networks

Assuming that our dataset $\mathcal{D}^{\text{avail}}$ is representative for all values of interest $\mathbf{y}_{t,\mathbf{s}}$, that is, that samples from $\mathcal{D}^{\text{avail}}$ are identically and independently distributed with some probability distribution $P(\mathbf{x}_{t,\mathbf{s}}, \mathbf{y}_{t,\mathbf{s}})$, allows us to learn a functional relationship $f : \mathcal{X} \to \mathcal{Y}$ using gradient descent algorithms. Our goal is then to generalize as well as possible on data

points that are not in $\mathcal{D}^{\text{avail}}$. That is, we want to perform well on spatial, temporal and spatio-temporal prediction tasks. In practice, however, this is often an invalid assumption as our data distribution $P(\mathbf{x}_{t,\mathbf{s}}, \mathbf{y}_{t,\mathbf{s}})$ can be variant across both space and time. To use the prediction power of DL models and the efficiency of stochastic gradient descent algorithms for training these, we have to design a learning method that is able to tackle such distribution shifts. Here, we consider modular neural network prediction models. For each individual application and its feature types, a large variety of architectures can be sensible. The architectures that we consider require a multi-input structure with at least one separate input for each feature type $\mathbf{x}_t$, $\mathbf{x}_\mathbf{s}$ and $\mathbf{x}_{\mathbf{s}t}$, and at least one being available for the given prediction task.

Figure 2 shows the general architecture of this type that we introduce as a spatio-temporal embedding network. Modules of this embedding network can be used to shape encoders that embed features into a vector space of arbitrary dimension. The last layer of each encoder is referred to as an embedding layer. Let $N_{(e)} \in \mathbb{Z}^+$ with $(e) \in \{\text{t, s, st, x}\}$ be the dimension of the vectors into which $\mathbf{x}_{t,\mathbf{s}}$ can be embedded, that is, the number of nodes of each embedding layer, and $\mathcal{U}^{(i)}$ be the sets of all embedded vectors in their respective vector spaces with $(i) \in \{\text{time, space, space–time, joint}\}$. With the proposed network architecture, we can then define the prediction model f_{NN} and its feature encoders $\text{enc}_{(i)}$ as functions

$$f_{NN} : \mathcal{X} \rightarrow y$$

$$\text{enc}_{(i)} : \mathcal{X} \rightarrow \mathcal{U}^{(i)}$$

The data selection problem

Given $\mathcal{D}^{\text{avail}}$, there usually exists a much larger set of data points that are not available to us. We refer to these as the candidate features $\mathcal{X}^{\text{cand}}$ and their complement labels y^{cand} which together shape the set of candidate data points $\mathcal{D}^{\text{cand}} = (\mathcal{X}^{\text{cand}}, y^{\text{cand}})$. Our goal is to choose the most informative subset of labels $y^{\text{choice}(*)}$ from the large pool of candidate labels y^{cand} such that our overall generalization error decreases the most, without exceeding a given number of labels, which we refer to as our data budget n_{budget}. During the data selection process, we assume that we have complete access to all existing features $\mathcal{X}$, the available labels y^{avail}, but not to any labels from y^{cand}. It hence has to hold that

$$(\mathcal{D}^{\text{avail}} \cup \mathcal{D}^{\text{cand}}) \subseteq \mathcal{D},$$

$$\mathcal{D}^{\text{avail}} \cap \mathcal{D}^{\text{cand}} = \varnothing,$$

$$y^{\text{choice}(*)} \subset y^{\text{cand}}.$$

The subset of labels that we eventually query without prior information about their values is likely to deviate from the optimal subset $y^{\text{choice}(*)}$. We refer to the actually queried subset of labels with y^{choice}. The feature–label pairs of queried data points are respectively referred to as $\mathcal{D}^{\text{choice}} = (\mathcal{X}^{\text{choice}}, y^{\text{choice}})$. One way to query labels is to do so one by one. Another, computationally more efficient way to do this is to use batches of data queries, particularly because we also train our neural network models with batches of data points between each ADL iteration. We define the batch size, or number of labels, that are queried in each step of an ADL process as $n_{\text{batch}} \in \mathbb{Z}^+$ and the total number of data selection steps as $n_{\text{iter}} \in \mathbb{Z}^+$. It hence has to hold that

$$n_{\text{iter}} n_{\text{batch}} \leq n_{\text{budget}}.$$

Embedded feature vectors

Given any of the encoders $\text{enc}_{(i)}$ that f_{NN} incorporates, we can encode each feature vector $\mathbf{x}_{t,\mathbf{s}}$, and single parts of it ($\mathbf{x}_t$, $\mathbf{x}_\mathbf{s}$ or $\mathbf{x}_{\mathbf{s}t}$), into their embedded vector spaces. We expect the distances of these vectors to each other to become increasingly meaningful in the context of our overarching prediction task as we train the actual prediction model f_{NN} (refs. [31–35]). As our encoders are modules of our prediction model, they are automatically trained each time we apply backpropagation on f_{NN} through stochastic gradient descent. Mutual information between parts of our feature vector can propagate back into each encoder, such that encoded parts of our feature vector can preserve information about all features. As a result, our method becomes sparse. For every feature vector $\mathbf{x}_{t,\mathbf{s}}$ and $(i) \in \{\text{time, space, space–time, joint}\}$, we can write

$$\forall \mathbf{x}_{t,\mathbf{s}} = \begin{pmatrix} \mathbf{x}_t \\ \mathbf{x}_\mathbf{s} \\ \mathbf{x}_{\mathbf{s}t} \end{pmatrix} \in \mathcal{X} \exists \{\hat{\mathbf{x}}_{(j)} = \text{enc}_{(i)}(\mathbf{x}_{(j)})\}_{(j)=\{(t,\mathbf{s}),t,\mathbf{s},\mathbf{s}t\}}.$$

We refer to predicted labels of data points as

$$\hat{\mathbf{y}}_{t,\mathbf{s}} = f_{NN}(\mathbf{x}_{t,\mathbf{s}}).$$

Clusters of embedded feature vectors

Given a set of vectors of the same dimension, we can calculate clusters based on the distances of these vectors to each other using algorithms such as *K*-means or affinity propagation. To execute most clustering algorithms, we need to determine the number of desired clusters or a minimum distance of members beforehand. To avoid assumptions regarding common distances in the embedded vector spaces, we only consider clustering methods that require a definition of the number of clusters beforehand. We refer to the number of clusters that we set for performing any of these clustering methods with n_{clusters}. For a clustering of embedded vectors to be valid, it hence has to hold that

$$n_{\text{clusters}} < |\mathcal{U}^{(i)}|,$$

and for data queries to be sensible furthermore that

$$n_{\text{clusters}} \ll |\mathcal{U}^{(i)}|.$$

After clustering the elements of any embedded vector set $\mathcal{U}^{(i)}$ with $(i) \in \{\text{time, space, space–time, joint}\}$, we get a first set of vectors $\mathbf{c}_l^{(i)}$ which describe the centre of each cluster with $l = 1 \ldots n_{\text{clusters}}$, and a set of values $m_k^{(i)}$ which describe the cluster membership identifiers (IDs) through an integer number for each clustered data point with

$$k = 1 \ldots |\mathcal{U}^{(i)}|.$$

Embedding uncertainty: distance of features to cluster centres

The distance between any two vectors of the same dimension can be calculated using inner products through, for example, kernel functions or other distance measures such as the cosine similarity. Using any of these measures, we can calculate the distance $d_{c,(j)}$ of every embedded vector $\hat{\mathbf{x}}_{(j)}$ to its corresponding cluster centre $\mathbf{c}^{(i)}_{m_k^{(i)}}$ with $(i) \in \{\text{time, space, space–time, joint}\}$, $(j) \in \{t, \mathbf{s}, \mathbf{s}t, (t, \mathbf{s})\}$ and $k = 1, \ldots, |\mathcal{U}^{(i)}|$ being the element ID that corresponds to the point $(t, \mathbf{s})$. We use this distance as a metric of uncertainty, called the embedding uncertainty of our ADL method. Alternatively, we can also use a Gaussian mixture model to cluster embedded feature vectors and express uncertainties in a single step.

The distance metric we use for both clustering candidates and computing their embedding uncertainty depends on both the embedding dimensions that we choose and the ratio between our query batch size and the size of our candidate dataset. With larger embedding

dimensions, the volume and as a result also the sparsity of our vector space increase quickly. This requires an increasing candidate data volume compared with the number of data points we choose in each ADL iteration to make informative queries. Further, the relative contrast, that is, the maximum distance between embedded vectors, becomes smaller in higher dimensions, which must be accounted for with higher numerical accuracy. We can, for instance, assume that fractional distance metrics increase the relative contrast of our embedding space and therefore lead to more stable data queries.

Batch ADL algorithm

Given our spatio-temporal embedding network and our embedding uncertainty $d_{c,(j)}$, we create a pool-based ADL method that queries a batch of labels $\mathbf{y}_{t,\mathbf{s}} \in y^{\text{cand}}$ from the candidate data pool in each iteration. Algorithm 1 provides the pseudo-code for this method, which we go through in more detail next.

Starting with a DL model f_{NN} that is trained on randomly chosen initial data (step 1), we can choose which feature type and corresponding encoder we want to use for querying candidate data points (step 2). Given some data budget n_{budget} and a maximum number of iterations n_{iter}, we create a data counter c_{budget} and an iteration counter c_{iter} that we set to zero and leave the set of queried data points empty before performing ADL (step 3). We start our ADL iterations by encoding each candidate data point (step 4.1). If the set of candidate data points is too large for this to be computationally feasible, we can sample a subset of candidate data points at random. Next, we cluster all embedded feature vectors with the number of clusters being equal to the number of candidates that we want to query (step 4.2). We can instead reduce the number of clusters and query more than one point per cluster to reduce the computational complexity. We then compute the distances to their respective cluster centres (step 4.3). We can then pick the most informative data point from each cluster using one of our ADL variants (step 4.4). The chosen subset of data points is then used for training our prediction model f_{NN} (step 4.5). Here, arbitrary techniques such as weight regularization, early stopping and adaptive learning rates can be used to enhance training. Next, we remove queried data points from the candidate data pool at a rate $\delta \in \mathbb{R}$ with $0 \leq \delta \leq 1$ (step 4.6). Here, the rate δ is the probability with which we remove a queried data point. A value of $\delta = 1$ means that all queried data points are removed, while a value of $\delta = 0$ means that all queried data points are kept. Before we continue with the next iteration, we update the set of queried candidates (step 4.7) and increment our data point counter by the number of newly queried labels among the chosen data points (step 4.8) and our iteration counter by one (step 4.9).

We can highlight two major differences from existing ADL methods. First, we remove queried data points from the candidate dataset at some rate δ. This allows us to re-use data points so as to explicitly reduce ($\delta \to 1$) or increase ($\delta \to 0$) a bias towards already queried data points. Second, we set the number of clusters of our candidates equal to the batch size of data points that we want to query in each iteration. This allows us to implicitly sample more points by building more clusters where data points are densely populated, hence having a sufficiently representative sample of our entire data population. Simultaneously, this allows us to cope with imbalanced data as we also create clusters where data are located in isolation and are available in small amounts in the encoded space.

The memory and time complexity of our algorithm depend on the size of the candidate data pool $|\mathcal{D}^{\text{cand}}|$. They can hence be reduced at the cost of less informative data queries by down-sampling the candidate data pool, or by querying more than one data point per cluster in each iteration. Another useful heuristic is to query candidates on the basis of embedded coordinates in time through the ADL variable $\mathbf{x}_t$ or in space through the ADL variable $\mathbf{x}_\mathbf{s}$. This reduces the number of candidates in the embedded vector space by a factor equal to the respective complementary coordinate as compared with using any other ADL variable that encodes features into unique coordinates in space–time. The time complexity of our algorithm further depends on the method we use to cluster embedded feature vectors (step 4.2), the method we use to compute the distance of candidate data to cluster centres (step 4.3) and the complexity of training our DL model (step 4.5). For most prediction tasks, it is realistic to assume that the complexity of step 4.5 is smaller than that of steps 4.2 and 4.3. If we, for instance, use the *K*-means++ algorithm to cluster embedded feature vectors and calculate the Manhattan distance between embedded vectors and their cluster centre, it holds that

$$\mathcal{O}\left(|\mathcal{D}^{\text{cand}}|N_{(e)}\log\left(N_{(e)}\right)\right) < \mathcal{O}\left(|\mathcal{D}^{\text{cand}}|N_{(e)}n^{\text{batch}}\right).$$

The computational complexity of our algorithm then breaks down to that of our clustering method (step 4.2), if we further make the realistic assumption that

$$\log(N_{(e)}) \ll n^{\text{batch}}.$$

Algorithm 1. A pseudo-code of the proposed batch ADL method.

1. Train f_{NN} on D^{avail}.
2. Choose $(j) \in \{t, \mathbf{s}, \mathbf{st}, (t,\mathbf{s})\} \leftrightarrow (i) \in \{time, space, space\text{-}time, joint\}$.
3. $c_{budget} = 0; c_{iter} = 0; \mathcal{D}^{choice} = \varnothing$.

while $c_{budget} < n_{budget}$ *and* $c_{iter} \leq n_{iter}$ **do**
  if $|\mathcal{X}^{cand}| >> n_{budget}$ **then**
    $\mathcal{X}^{cand} \leftarrow \mathcal{X}^{subset} \subset \mathcal{X}^{cand}$
  end
  4.1 $\hat{\mathbf{x}}_{(j)} = enc_{(i)}(\mathbf{x}_{(j)}) \forall \mathbf{x}_{t,\mathbf{s}} \in \mathcal{X}^{cand}$
  4.2 Cluster $\hat{\mathbf{x}}_{(j)}$ with $n_{clusters} = n_{batch}$.
  4.3 Compute $d_{c,(j)}$.
  4.4 $\mathcal{Y}^{choice}_{(c_{iter})} = \{\hat{\mathbf{x}}_{(j)}\}_{k=1}^{n_{batch}}$.
  4.5 Train f_{NN} on $\mathcal{D}^{choice}_{(c_{iter})} = (\mathcal{X}^{choice}_{(c_{iter})}, \mathcal{Y}^{choice}_{(c_{iter})})$.
  4.6 $\mathcal{D}^{cand} \leftarrow \mathcal{D}^{cand} \setminus \mathcal{D}^{choice}_{(c_{iter})}$ at rate δ
  4.7 $\mathcal{D}^{choice} \leftarrow \mathcal{D}^{choice} \cup \mathcal{D}^{choice}_{(c_{iter})}$
  4.8 $c_{budget} \leftarrow |\mathcal{D}^{choice}|$
  4.9 $c_{iter} \leftarrow c_{iter} + 1$
end

Datasets

We are given the electric consumption measurements of 100 and 400 buildings in Switzerland in 15 min steps from local distribution system operators. Using the geographic coordinates of these buildings, we further collect aerial imagery of each building with a resolution of 25 cm per pixel[36]. We then cluster all buildings that are in a distance of at most 1 km to each other. For each cluster of buildings, we calculate the cluster centres and collect a total of nine meteorological time series measurements from reanalysis data for each of these clusters with 1 h accuracy[37,38]. The meteorological values that we use consist of air density in kg m^{-3}, cloud cover, precipitation in mm h^{-1}, ground-level solar irradiance in W m^{-2}, top of atmosphere solar irradiance W m^{-2}, air temperature in °C, snowfall in mm h^{-1}, snow mass in kg m^{-2} and wind speed.

We predict the next 24 h of electricity consumption. For each of the nine meteorological conditions, we consider a historical time window of 24 h at the same time which results in a dimension of 216 (nine times 24) for space-time variant features. Time stamps are ordinal encoded and contain information about the month, day, hour and quarter-hour in which the corresponding electricity consumption of a building occurs. We encode each element of our time stamps separately, which gives us a time-variant feature dimension of four. Images of buildings are processed using histograms of their pixel values with 100 bins for each image channel (red, green and blue) which results in a dimension of 300. We use histograms instead of original images in order to preserve the privacy of our data sources and experimentally find that 100 bins preserve sufficient information from the

original images. These choices can, however, be improved through hyperparameter optimization. We can hence set the dimensions of the feature and label vectors to

$$D_{\mathrm{t}} = 4,$$

$$D_{\mathrm{s}} = 300,$$

$$D_{\mathrm{st}} = 216,$$

$$D_{\mathrm{x}} = D_{\mathrm{t}} + D_{\mathrm{s}} + D_{\mathrm{st}} = 520,$$

$$D_{\mathrm{y}} = 96.$$

Training, validation and testing data

Given a number of data points that are available to us, we create training, validation and testing data for our hypothesis test. The training data are used to fit our prediction model before performing ADL. The validation data are used to avoid overfitting our model to the training data through early stopping. The testing data are used as the candidate data pool on which we perform ADL to train our prediction model. We separate our testing data into spatial, temporal and spatio-temporal prediction tests. We use 4.5% of our data for initial training, 4.5% for validation and 91% for testing. We further split our testing data such that 23% of them represent spatial predictions, another 23% temporal predictions and 54% spatio-temporal predictions. In the following, we refer to training, validation and testing data with $\mathcal{D}^{\mathrm{train}}$, $\mathcal{D}^{\mathrm{val}}$ and $\mathcal{D}^{\mathrm{test}}$.

Prediction model and feature encoders

We construct our DL model from multiple subnetworks. The network which processes meteorological data consists of a one-dimensional convolutional neural network layer with 16 filters. The networks which process time stamp data and the histograms of building image pixels each contain one densely connected hidden layer with 1,000 nodes. The joint encoder concatenates the outputs of each of these networks and adds another densely connected hidden layer with 1,000 nodes. All embedding layers consist of 100 nodes. We experimentally find that embedding dimensions of 10–100 work well for our candidate data size and the distance metrics we use. The prediction model then takes the output of the joint encoder and adds another layer of 1,000 densely connected nodes before mapping the joint inputs to the desired output with 96 densely connected nodes. Our model contains 10,744,600 trainable and 0 non-trainable parameters. These choices can, however, be improved through neural architecture search. For the encoder outputs of all $(e) \in \{\mathrm{t}, \mathrm{s}, \mathrm{st}, \mathrm{x}\}$, we can write

$$N_{(e)} = 100.$$

Loss function

We use the mean squared error, also known as the L2 loss, between predicted labels $\hat{\mathbf{y}}_{t,\mathbf{s}}$ and true labels $\mathbf{y}_{t,\mathbf{s}}$ as our loss $L(\hat{\mathbf{y}}_{t,\mathbf{s}}, \mathbf{y}_{t,\mathbf{s}})$. One can equivalently use the mean absolute error, also known as the L1 loss, or variations from these with minor impacts on the empirical results. In each epoch of training and validation, as well as for each test, we calculate the total loss function $\mathrm{loss}(\mathcal{D}^{(d)})$ as the average loss of all data points in the respective datasets $D^{(d)}$, where $(d) \in \{\mathrm{train}, \mathrm{val}, \mathrm{test}\}$. With $j = 1, \ldots, |\mathcal{D}^{(d)}|$ being the jth element that corresponds to the point $(t, \mathbf{s})$ in $\mathcal{D}^{(d)}$, we can write for all pairs of $(\hat{\mathbf{y}}_j, \mathbf{y}_j) \in \mathcal{D}^{(d)}$ that

$$L(\hat{\mathbf{y}}_{t,\mathbf{s}}, \mathbf{y}_{t,\mathbf{s}}) = \frac{\sum_{k=1}^{D_y} (y_{t,\mathbf{s},k} - \hat{y}_{t,\mathbf{s},k})^2}{D_y}$$

and

$$\mathrm{loss}(\mathcal{D}^{(d)}) = \frac{\sum_{j=1}^{|\mathcal{D}^{(d)}|} L(\hat{\mathbf{y}}_j, \mathbf{y}_j)}{|\mathcal{D}^{(d)}|}.$$

Experiments

We assume our data budget to be 50% of the size of our candidate data pool. That is, we want to choose the more informative half of candidate data points. We perform ten iterations of the above Algorithm 1 where we query 10% of our data budget in each iteration. We train our prediction model for 30 epochs and use an early stopping patience of 10 epochs when training our prediction model on the initially available data and in each iteration of Algorithm 1. We can write

$$n_{\mathrm{budget}} = 0.5|\mathcal{D}^{\mathrm{cand}}|,$$

$$n_{\mathrm{iter}} = 10,$$

$$n_{\mathrm{clusters}} = n_{\mathrm{batch}} = 0.1 n_{\mathrm{budget}} = 0.05|\mathcal{D}^{\mathrm{cand}}|.$$

We use the K-means++ algorithm to cluster embedded feature vectors, and the Laplacian kernel to calculate the distance between each embedded feature vector and its cluster centre. Given the embedded vector set $\mathcal{U}^{(i)}$ with $(i) \in \{\mathrm{time}, \mathrm{space}, \mathrm{space{-}time}, \mathrm{joint}\}$, the vectors that describe the centre of each cluster $\mathbf{c}_l^{(i)}$ with $l = 1 \ldots n_{\mathrm{clusters}}$ and a cluster membership number $m_k^{(i)}$ for each embedded feature $k = 1 \ldots |\mathcal{U}^{(i)}|$, we calculate the distance $d_{\mathrm{c},(j)}$ for every feature type $(j) \in \{t, \mathbf{s}, \mathbf{s}t, (t, \mathbf{s})\}$, corresponding encoder outputs $(e) \in \{\mathrm{t}, \mathrm{s}, \mathrm{st}, \mathrm{x}\}$ and point in time–space $(t, \mathbf{s})$ as

$$d_{\mathrm{c},(j)} = \exp\left(-\frac{||\hat{\mathbf{x}}_{(j)} - \mathbf{c}^{(i)}_{m_k^{(i)}}||_1}{N_{(e)}}\right).$$

We test Algorithm 1 for every partial feature vector, and the entire feature vector separately. Since our labels have a similar dimension ($D_{\mathrm{y}} = 96$) to our embedded features ($N_{(e)} = 100$), we use the predicted labels ($\hat{\mathbf{y}}_{t,\mathbf{s}}$) as our jointly embedded feature vectors ($\mathbf{x}_{t,\mathbf{s}}$). We can also design our joint feature encoder to contain all layers of our DL prediction model except for the output layer. This is proposed in ref. [39] and represents a special case of the method we propose. We also evaluate a scenario in which we query candidate data points on the basis of the distance of their true labels $\mathbf{y}_{t,\mathbf{s}}$. We conduct our tests for the two cases that we mainly distinguish: first, we remove each queried point from the candidate data pool at the end of each ADL iteration ($\delta = 1$); second, we keep queried data points in the candidate pool throughout all ADL iterations ($\delta = 0$). We conduct additional experiments for reducing the time complexity of Algorithm 1. We subsample our candidate data pool to up to 25% or reduce the number of clusters to up to 25% of our queried batch size in each iteration. Furthermore, we query candidates on the basis of their embedded spatial features/coordinate only.

Compute environment

Each experiment is run with a dedicated graphical processing unit. We further use two central processing units with 64 cores each, mainly for calculating clusters in our candidate data pool in parallel. The compute time for assessing the informativeness of each candidate data point can hence be reduced when increasing the number of central processing unit cores in each experiment.

Data availability

All of our data and results can be accessed on the Harvard Dataverse under a CC0 1.0 license (https://doi.org/10.7910/DVN/3VYYET). For privacy

maintenance, we only provide load profiles that are sampled from the original data and histograms of pixel values of building images, which can be used to reproduce all the elements of our original experiments.

Code availability

All results, figures and tables can be reproduced using step-by-step instructions in Jupyter notebook sessions that we provide in a public Github repository (https://github.com/ArsamAryandoust/DataSelectionMaps). We further maintain a Python package (https://pypi.org/project/altility) and a Docker container (https://hub.docker.com/r/aryandoustarsam/altility) implementation of our algorithm. All code is available under an MIT license.

References

1. Patt, A. *Transforming Energy – Solving Climate Change with Technology Policy* (Cambridge Univ. Press, 2015).
2. *IPCC Special Report on Global Warming of 1.5 °C* (eds Masson-Delmotte, V. et al.) (WMO, 2018).
3. *IPCC Climate Change 2021: The Physical Science Basis* (eds Masson-Delmotte, V. et al.) (Cambridge Univ. Press, 2021).
4. Hahn, H., Meyer-Nieberg, S. & Pickl, S. Electric load forecasting methods: tools for decision making. *Eur. J. Oper. Res.* https://doi.org/10.1016/j.ejor.2009.01.062 (2009).
5. Soliman, A.-h. S. & Al-Kandari, A. M. *Electric Load Forecasting* (Butterworth-Heinemann, 2010).
6. Alfares, H. K. & Nazeeruddin, M. Electric load forecasting: literature survey and classification of methods. *Int. J. Syst. Sci.* **33**, 23–24 (2002).
7. Kofi Nti, I., Teimeh, M., Nyarko-Boateng, O. & Adekoya, A. F. Electricity load forecasting: a systematic review. *J. Electr. Syst. Inform. Technol.* **7**, 13 (2020).
8. Shi, J., Liu, Y. & Yu, N. Spatio-temporal modeling of electric loads. *IEEE* https://doi.org/10.1109/NAPS.2017.8107311 (2017).
9. Tascikaraoglu, A. Evaluation of spatio-temporal forecasting methods in various smart city applications. *Renew. Sustain. Energy Rev.* **82**, 424–435 (2018).
10. Severiano, C. A., Cândido de Lima eSilva, P., Cohen, M. W. & Gadelha Guimarãesae, F. Evolving fuzzy time series for spatio-temporal forecasting in renewable energy systems. *Renew. Energy* **171**, 764–783 (2021).
11. Willis, H. L. *Spatial Electric Load Forecasting* (Marcel Dekker, 2002).
12. Rolf, E. et al. A generalizable and accessible approach to machine learning with global satellite imagery. *Nat. Commun.* **12**, 4392 (2021).
13. Burke, M., Driscoll, A.Lobell, D. B. & Ermon, S. Using satellite imagery to understand and promote sustainable development. *Science* https://doi.org/10.1126/science.abe8628 (2021).
14. Melo, J. D. & Carreno, E. M. *Data Issues in Spatial Electric Load Forecasting* (IEEE, 2014).
15. Milam, M. & Venayagamoorthy, G. K. Smart meter deployment: US initiatives. *IEEE* https://doi.org/10.1109/ISGT.2014.6816507 (2014).
16. Sovacool, B. K., Hook, A., Sareen, S. & Geels, F. W. Global sustainability, innovation and governance dynamics of national smart electricity meter transitions. *Glob. Environ. Change* https://doi.org/10.1016/j.gloenvcha.2021.102272 (2021).
17. Kezunovic, M., Xie, L. & Grijalva, S. The role of big data in improving power system operation and protection. *IEEE* https://doi.org/10.1109/IREP.2013.6629368 (2013).
18. Yu, N. et al. The role of big data in improving power system operation and protection. *IEEE* https://doi.org/10.1109/ISGT.2015.7131868 (2015).
19. Stein, A. L. Artificial intelligence and climate change. *Yale J. Regul.* **37**, 890–934 (2020).
20. Rolnick, D. et al. Tackling climate change with machine learning. *ACM Comput. Surv.* https://doi.org/10.1145/3485128 (2022).
21. Kuo, P., Liang, D., Gao, L. & Lou, J. Probabilistic electricity price forecasting with variational heteroscedastic Gaussian process and active learning. *Energy Convers. Manage.* **89**, 298–308 (2015).
22. Wang, Z., Zhao, B., Guo, H., Tang, L. & Peng, Y. Deep ensemble learning model for short-term load forecasting within active learning framework. *Energies* https://doi.org/10.3390/en12203809 (2019).
23. Zhang, L. & Wen, J. Active learning strategy for high fidelity short-term data-driven building energy forecasting. *Energy Build.* https://doi.org/10.1016/j.enbuild.2021.111026 (2021).
24. Kuster, C., Rezgui, Y. & Mourshed, M. Electrical load forecasting models: a critical systematic review. *Sustain. Cities Soc.* **35**, 257–270 (2017).
25. Panamtash, H., Mahdavi, S., Dimitrovski, A. & Zhou, Q. *Comparison of Probabilistic Forecasts for Predictive Voltage Control* (North American Power Symposium, 2021).
26. Chen, T. et al. Big self-supervised models are strong semi-supervised learners. In *Proc. of the 34th Conference on Advances in Neural Information Processing Systems* 33 (eds Larochelle, H. et al.) 22243–22255 (NeurIPS, 2020).
27. Yue, X. et al. Prototypical cross-domain self-supervised learning for few-shot unsupervised domain adaptation. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13834–13844 (CVF, 2021).
28. Yang, W. et al. Class distribution alignment for adversarial domain adaptation. Preprint at https://arxiv.org/abs/2004.09403 (2020).
29. Saito, K., Saenko, K. & Liu, M.-Y. COCO-FUNIT: few-shot unsupervised image translation with a content conditioned style encoder. In *Proc. of 16th European Conference on Computer Vision* (eds Vedaldi, A. et al.) 382–398 (Springer, 2020).
30. Jain, A., Zamir, A. R., Savarese, S. & Saxena, A. Structural-RNN: deep learning on spatio-temporal graphs. In *2016 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2016).
31. Mikolov, T., Chen, K., Corrado, G. & Dean, J. Efficient estimation of word representations in vector space. Preprint at https://arxiv.org/abs/1301.3781 (2013).
32. Frome, A. et al. DeViSE: a deep visual-semantic embedding model. In *Proc. of the 26th Advances in Neural Information Processing Systems* 26 (eds Burges, C.J. et al.) 2121–2129 (NIPS, 2013).
33. Pennington, J., Socher, R. & Manning, C. D. GloVe: Global vectors for word representation. In *Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing* (eds Moschitti, A. et al.) 1532–1543 (ACL, 2014).
34. Perozzi, B., Al-Rfou, R. & Skiena, S. DeepWalk: online learning of social represenations. Preprint at https://arxiv.org/abs/1403.6652 (2014).
35. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. Preprint at https://arxiv.org/abs/1810.04805 (2019).
36. GeoVITE – user-friendly geodata service. *Swiss Federal Office of Topography* https://geovite.ethz.ch/ (2020).
37. Pfenninger, S. & Staffel, I. Long-term patterns of European PV output using 30 years of validated hourly reanalysis and satellite data. *Energy* **114**, 1251–1265 (2016).
38. Staffel, I. & Pfenninger, S. Using bias-corected reanalysis to simulate current and future wind power output. *Energy* **114**, 1224–1239 (2016).
39. Ash, J. T., Zhang, C., Krishnamurthy, A., Langford, J. & Agarwal, A. Deep batch active learning by diverse, uncertain gradient lower bounds. In *Proc. of International Conference on Learning Representations* https://openreview.net/forum?id=ryghZJBKPS (2020).

Acknowledgements
We acknowledge funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 837089 (A.A., A.P. and S.P.), for the SENTINEL project.

Author contributions
A.A. conceptualized the research, designed and implemented all the methods, conducted all the analyses and drafted the manuscript. S.P. edited and revised the manuscript and reviewed the code. A.P. helped in interpreting the data and supervised the creation of the manuscript.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42256-022-00552-x.

Correspondence and requests for materials should be addressed to Arsam Aryandoust.

Peer review information *Nature Machine Intelligence* thanks Valentin Robu and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.