Google

Performance Prediction for Large Systems via Text-to-Text Regression

Yash Akhauri[*1,2], **Bryan Lewandowski**[1], **Cheng-Hsi Lin**[1], **Adrian N. Reyes**[1], **Grant C. Forbes**[3], **Arissa Wongpanich**[1], **Bangding Yang**[1], **Mohamed S. Abdelfattah**[2], **Sagi Perel**[1] **and Xingyou Song**[1]
[1]Google, [2]Cornell University, [3]North Carolina State University

*Work performed as a student researcher at Google Research.
Code: https://github.com/google-deepmind/regress-lm

In many industries, predicting metric outcomes of large systems is a fundamental problem, driven largely by traditional tabular regression. However, such methods struggle on complex systems data in the wild such as configuration files or system logs, where feature engineering is often infeasible. We propose text-to-text regression as a general, scalable alternative. For predicting resource efficiency on Borg, Google's massive compute cluster scheduling system, a 60M parameter encoder-decoder, trained from random initialization, achieves up to a near perfect 0.99 (0.9 average) rank correlation across the entire fleet, and 100x lower MSE than tabular approaches. The model also easily adapts to new tasks in only 500 few-shot examples and captures the densities of complex outcome distributions. Ablation studies highlight the importance of using encoders, increasing sequence length, and the model's inherent uncertainty quantification. These findings pave the way for universal simulators of real-world outcomes.

1. Introduction

Performance prediction has been an important problem for various industrial system use cases, ranging from latency prediction (Madhyastha et al., 2006), execution times (Venkataraman et al., 2016), scheduling conflicts (Manousis et al., 2020), and transaction response timing (Stewart et al., 2007). While traditional methods have significantly relied on expert domain knowledge to model specific numeric metrics, more recent data-centric techniques have predominantly used machine-learning based regression methods by training models which predict metric y given a set of features x.

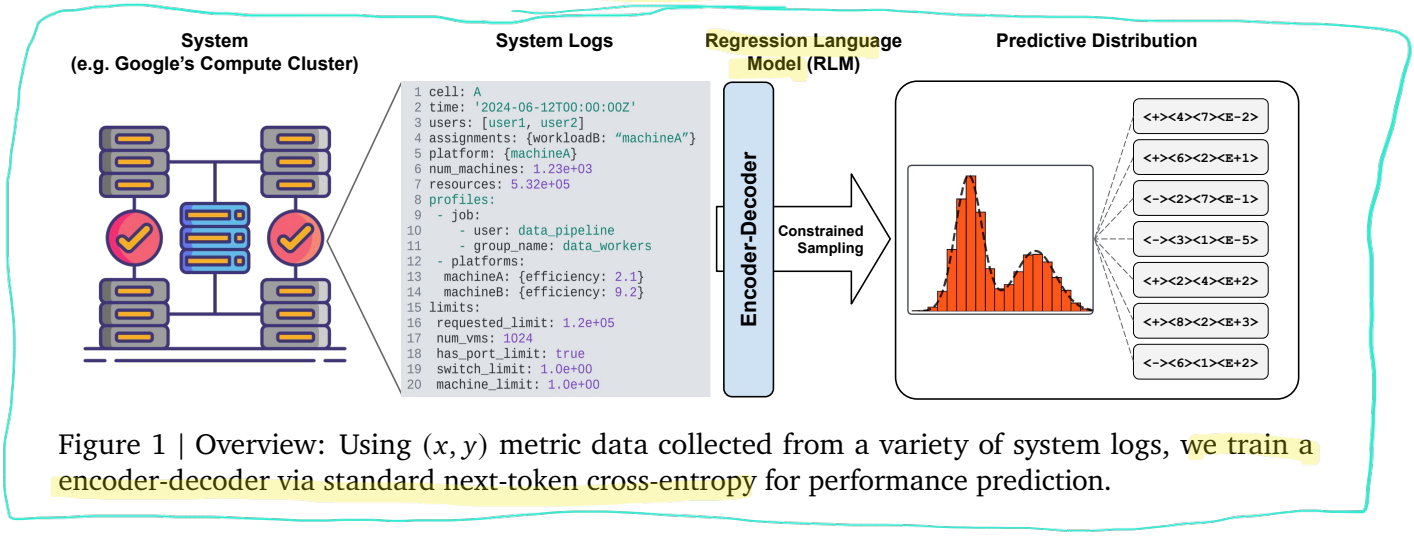

Figure 1 | Overview: Using (x, y) metric data collected from a variety of system logs, we train a encoder-decoder via standard next-token cross-entropy for performance prediction.

However, standard machine learning regression techniques such as random forests and multi-layer perceptrons require the features x to be represented as flat fixed-length tensors, which can be difficult for certain prediction problems with highly non-tabular representations. Even if such feature engineering concerns are resolved, compressing features only as numbers without much context can

be highly lossy, and end up treating the system as a blackbox (Fu et al., 2021), which can drastically decrease predictive performance.

Graybox techniques (Didona et al., 2015) have attempted to mitigate these concerns by combining both domain-expertise and machine learning, by e.g. learning the coefficients to an already-derived symbolic expression (e.g. latency of a website as a linear function of users). These approaches often require large amounts of prior knowledge on the relationship between x and y, and are very restrictive in their applicability.

However, the advent of language models has since enabled *text-to-text regression*, which can avoid many of the issues regarding tensor-based feature engineering. Recently, Song et al. (2024a) showed that models even of relatively small size (<1B parameters), which we term as *Regression Language Models (RLMs)*, are capable of enormous transfer learning if trained over large amounts of (x, y) regression data, being able to improve prediction accuracy simply by absorbing information from multiple different regression tasks. Surprisingly, training with next-token prediction over numeric representations via cross-entropy loss is also quite sample efficient (Song and Bahri, 2025), capable of matching or even outperforming standard regression methods when given the same data.

In this paper, we demonstrate RLMs can be applied to even simulating efficiency outcomes by training over large amounts of data from large-scale industrial systems such as Google's "Borg" compute cluster, which can drastically reduce production costs. In summary, our contributions are as follows:

- RLMs are capable of predicting numeric outcomes, such as efficiency metrics of industrial systems (e.g. Google's entire compute cluster), with high precision over complex feature representations and multi-modal outcome distributions.
- By fine-tuning over very small amounts of new (x, y) training data, such models are highly capable of few-shot adaptation, wherein a pretrained model is able to remain highly accurate on unseen compute clusters or scenarios.
- Comprehensive ablations provide insights into the performance improvement effects of changing sequence length, model size, feature observability, architecture, learning rate, and early stopping, in addition to the model's natural uncertainty quantification abilities.

2. System Performance Prediction

2.1. Background

We provide a brief overview of Borg (Verma et al., 2015), the compute management system at Google. The system comprises of a centralized manager for scheduling jobs, and a list of machines located in different *cells* with compute resources. Each user may send in a *task request*, containing a binary executable with additional details such as resources needed, number of replicas, and so on. The role of the manager is to place each task on an appropriate machine (i.e., one where the requisite resources are available).

To effectively allocate computing tasks to resources, the scheduler must know the resource consumption (i.e. utilization) of a task on that machine and ultimately how much useful work can be done (i.e. productivity), in order to execute a specialized bin-packing algorithm for assigning jobs, to maximize total productivity across the entire compute cluster.

For a given process, this productivity metric is formally defined as Millions of Instructions Per Second per Google Computing Unit (Schneider and Mattia, 2024) of usage, or "MIPS per GCU". Intuitively, this metric represents the amount of work that gets done per unit of time by using per unit of computing resource. Empirically, *after* job assignment has been made, MIPS per GCU can be computed periodically

by profiling the total instructions executed and total CPU cycles consumed in a 10-second time window.

This metric can be affected by numerous factors, ranging from current memory and CPU usage, hardware type, and workload mixes across numerous machines. Furthermore, even variations in time can even have an effect, as workloads within a physical cell depend on the usage patterns by users within the same geographic location. Additionally, hyperparameters affecting the behavior of the underlying bin-packing algorithm may also strongly affect the outcome.

Fortunately, Google has developed a digital twin of Borg, a sophisticated backtesting framework which uses checkpoint files of real-world clusters to replicate the state of those clusters, perform the scheduling algorithm, and determine the aggregate cluster-wide MIPS per GCU across all machines. However, due to the inherently sequential nature, generating even a single outcome can require between 1 to 18 hours of computation regardless of resources used.

Fortunately, such outcomes are logged as valuable offline datasets for training cheap regression models, albeit from limited data. If a regressor is able to accurately predict this metric with negligible inference time, this can lead to large savings not just from avoiding computation, but also from faster optimization of the MIPS per GCU overall. For instance, Google Vizier (Golovin et al., 2017) is regularly used to tune the Borg scheduler's hyperparameters, but unfortunately its Gaussian Process regressor (Song et al., 2024b) can at most only observe tabular data formats, drastically limiting its predictive and overall optimization performance.

2.2. Prediction Task

The goal is to predict the MIPS per GCU efficiency metric, i.e. a single observed floating point number, after a specialized bin-packing algorithm has been performed, given the initial state of the compute cluster, time window used for collecting metric data, and hyperparameters of the scheduling algorithm. More specifically, the following available information may all be used as input features:

- Cluster name (i.e. "cell").
- Physical location of the cell.
- The window of time used to collect the state of the cell and the exact time the performance profiles were collected.
- Hyperparameters affecting the behavior of the Borg scheduler's bin-packing algorithm.
- Concurrent entities or teams using the largest amount of compute within this cell.
- Distribution and network hierarchy of hardware platforms installed in the cell.
- A list of job-on-machine performance profiling results.

All of these features are presumed to be important based on minimal domain knowledge, but it is not apriori clear how exactly they may affect this metric. For example, the efficiency metric is heavily dependent on the cell, which is the specific cluster of machines and hardware, but the actual cell name itself is not a true feature. Features such as timing windows and hyperparameter names can be shared or correlated across different tasks, but may affect efficiency metrics differently.

Furthermore, the prediction task can be dynamic, as new features values may appear over time, due to platform upgrades and hardware changes. The features may also contain varying types, e.g. categorical hyperparameters, different subsets of hardware types, and cyclical timestamps. Additional complexity arises due to the deep nested nature of many of the features – e.g. job-on-machine profilings containing information about jobs and which machines they utilize, which also contain information about hardware profiles, and so forth. Lastly, the metric may also possess various forms of noise, some dependent on the features observed by the regressor, as we discuss below.

2.3. Observed Features vs. Uncertainty

The most broad abstraction to define our prediction task is to assume there is an underlying distribution $p(y|x)$ where x is the *full state* of the world. Thus the randomness induced by $p(y|x)$ can be considered irreducible noise, also known as *aleatoric uncertainty* (Hüllermeier and Waegeman, 2019), which limits the optimal performance of a pointwise regressor. For Borg, this can come from the inherent randomness of the bin-packing algorithm and stochastic load/demand (e.g. for user-facing services).

More formally, based on the well-known *Bias-Variance decomposition*, the expected squared error of a pointwise regressor f_θ per input x is lower bounded by the variance of the y:

$$\mathbb{E}_{y\sim p(y|x)}\left[(y - f_\theta(x))^2\right] \geq \text{Var}(y|x) \tag{1}$$

However, if a regressor is only able to observe a partial subset or limited representation $\phi(x)$ of the full state, it will be unable to distinguish separate x, x' if $\phi(x) = \phi(x')$. This lack of distinguishability induces *epistemic uncertainty*, and instead leads to an even higher right hand variance term $\text{Var}(y|\phi(x))$, limiting the regressor's optimal performance even more (theory in Appendix A.1). For practical purposes, given an offline test dataset $\mathcal{D} = \{(x_i, y_i)\}_{i\geq 0}$, we can estimate $\text{TotalVariance}_\phi(\mathcal{D})$ to provide lower bounds on the mean squared error (MSE) of regression methods evaluated over all test data, as:

$$\text{TotalVariance}_\phi(\mathcal{D}) = \frac{1}{K}\sum_{k=1}^{K}\text{Var}(y|x \in \mathcal{X}_k) \tag{2}$$

where $\mathcal{X}_1, \ldots, \mathcal{X}_K$ are equivalence classes partitioning $\mathcal{D}$, i.e. $x, x' \in \mathcal{X}_k$ iff $\phi(x) = \phi(x')$, and variance is calculated empirically over the set of all y-values obtained from inputs within $\mathcal{X}_k$. Note that if ϕ is "null" (no observed features), this expression would simply be the variance of the entire y-population.

Similar bounds exist for other regression-based metrics, e.g. rank correlations, since determining the relative rankings of y-values from the same x-equivalence class would be impossible, and a density estimator p_θ's log-likelihood would be bounded by the conditional entropy of $p(y|\phi(x))$. In any case, it thus would be ideal if the regressor is able to maximize the amount of features it observes in order to minimize epistemic uncertainty.

2.4. Why Text-based Regression?

Representing all of the features above into one single fixed-length tensor in $\mathbb{R}^d$ will be notoriously difficult if using a traditional tabular regressor such as a multi-layer perceptron (MLP) or random forest. Many of the features such as the list of jobs and platforms can contain arbitrary cardinalities per x, and would therefore need to be grouped using hand-selected methods or heuristics. Even if a useful set of hand-selected metrics can be found, tabular featurization requires apriori defining a finite number of classes per categorical parameter and maximum/minimum bounds for numbers for normalization – when a new class of machines or workloads emerges, the entire process must be started from scratch and all the training data produced from the prior method's rigid featurization are made incomplete and invalidated.

A flexible text-based input representation simply resolves these issues by allowing variable sequence length inputs and does not require explicit enumeration of categorical features nor normalization of continuous ones. By observing all available features, especially nested ones difficult to represent as tabular formats, the model minimizes epistemic uncertainty and opens the doors to achieving the best performance possible. Furthermore, such a text-based model would not need to restart from scratch when encountering new sources of data, but rather can be simply fine-tuned over new tasks to allow fast few-shot adaptation by transferring knowledge learned from previously trained checkpoints.

```
cell: cell_a
2024/06/02 17:00:00 PDT, Day:Fri Week:21
search_space:
  {'JOB/data_pipeline/PRODUCTION_WORKLOAD':
    ['machineE', 'machineA', 'none_selected']}
assignments: {"JOB/data_pipeline/PRODUCTION_WORKLOAD": "machineA"}
distributions:
- platform: {machineA}
  num_machines: 1.239e+03
  low_level_zones: 5.200e+01
  mid_level_zones: 5.200e+01
  high_level_zones: 4.300e+01
  resources: 5.481e+05
job_profiles:
- job: {user: data_pipeline, group_name: data_pipeline_workers}
  platform_profiles:
    machineD: {mean_mips_per_resource_usage: 8.165e+02}
    machineA: {mean_mips_per_resource_usage: 9.590e+02}
    machineF: {mean_mips_per_resource_usage: 8.321e+02}
    machineC: {mean_mips_per_resource_usage: 7.098e+02}
  limits:
    job_requested_resource_limit: 1.217e+04
    job_requested_num_vms: 1087
```

Figure 2 | Example anonymized string representations of some features used to construct x. More detailed representation can be found in Appendix B.1.

Feature Type	Average Character Count
Cell Name	3
Physical Location	8
Time Window	86
Scheduler Hyperparameters	1,082
Machine Distribution	461
Job-on-machine Performance	268,157

Table 1 | Average character counts for each YAML feature.

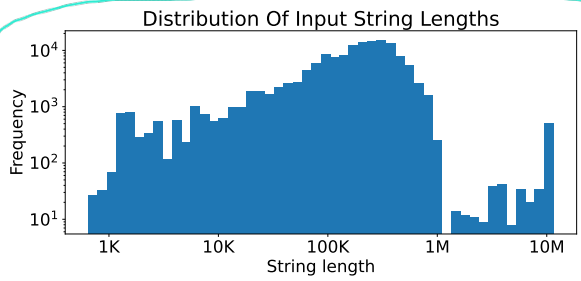

Figure 3 | Distribution of string lengths.

In Figure 2, we provide an example of a string, represented in a standard format (e.g. YAML), that can be sent to the model. Table 1 gives the average character counts for each of the features across all data, while Figure 3 gives the distribution of total string lengths.

3. Method

Below, we outline the technical details of our RLM, primarily based on OmniPred (Song et al., 2024a). For a reference survey on standard language model training and fine-tuning, we refer the reader to Zhang et al. (2023).

3.1. Preliminaries: Text-to-Text Regression

In the standard language model setting, given a batch of (prompt, response) pairs, one performs model updates by minimizing the next-token cross-entropy loss over response tokens. In the text-to-text regression case, the prompt corresponds to a string representation of x, while the response corresponds also to a textual or structured token representation of the floating value y. This training can be performed over *any* collection of data points, due to the universality of string representations. At inference time, the model can be interpreted as a density estimator $p_\theta(y|x)$, from which a pointwise prediction $\widehat{y}$ can be made by aggregating i.i.d. samples $y^{(1)}, \ldots, y^{(s)} \sim p_\theta(y|x)$.

3.2. Design Choices

Some of the following design choices for our RLM can be considered contrary to popular belief. We justify their choices below, some of which require nuanced discussion based on findings from prior work and this paper's ablations:

Decoding-based Output: We train with cross-entropy loss over tokens rather than error-based loss (e.g. MSE) over an additional value head. The cross-entropy loss magnitude is agnostic to the strength of the prediction gap, leading to more stable training over multiple different y-values, and avoids over-focusing on tasks which naturally have larger errors due to wider y-value spreads.

Furthermore, it has been broadly observed across both optimization (Tan et al., 2025) and reward modeling communities (Mahan et al., 2024; Zhang et al., 2025) that presumably due to their overly compressive nature, embedding-based or logit-based methods can perform worse than simply decoding the numeric prediction end-to-end.

Use of Encoder: Many current LLM designs only use the decoder-only architecture, as it simplifies designs by concatenating (prompt, response) pairs together without the need for specifying separate sequence lengths. However, as we show in our ablations in Section 5.2, separate encoder layers are necessary for processing complex x, and decoder layers alone are suboptimal.

No Language Pretraining: Despite the current trend of pretraining models on *human-generated text* in e.g. English, it is not necessary nor guaranteed beneficial to use a pretrained LLM checkpoint from which to train a regression model. In fact, Song et al. (2024a) found that tabular regression is possible *tabula rasa* with a randomly-initialized language model. This is presumably because regression only requires learning the correlations between different structured tokens and does not necessarily benefit from the semantic meaning behind words.

y-Tokenization: The cross-entropy loss does not have an inherent notion of numeric distance, as tokenization effectively discretizes the real number line. Since the model needs to learn an embedding for each token, it is important to minimize the vocabulary size used for representing floats, e.g. using <0>, <1>,..., <999> is far less effective than using a few digit tokens <0>, <1>, ..., <9>. We use the P10 tokenization found in (Charton, 2022), in which a y is represented using special sign, mantissa, and exponent tokens, e.g. <+><7><2><5><E-1> represents $725 \times 10^{-1} = 72.5$. This tokenization is also *normalization-free*, which allows easy multi-task training without needing to precompute minimum or maximum y-value bounds for every separate task.

Context-Free: Our model is a *context-free* regressor which only observes a single x and returns a single y, instead of first preprocessing multiple $(x_1, y_1), (x_2, y_2), \ldots$ in-context (Vacareanu et al., 2024). This maximizes use of the sequence length for observing the string representation of a single x, which may already be thousands of tokens long. This further allows *unlimited* data to be absorbed within the model weights, rather than having finite limits at inference time due to the context buffer. This distinction is analogous to using random forests and MLPs instead of Gaussian Processes for traditional regression, and weight-based updates instead of hidden memory states for meta-learning.

3.3. Fine-tuning

Similar to standard LLM practices, the RLM also allows fine-tuning against additional data even after pretraining. This serves two different purposes: (1) adaptation to a new unseen regression task using pretrained knowledge, and (2) refocusing over a previously seen task (e.g. if the pretraining dataset was too large).

To do so, we simply restore both the weights of a pretrained checkpoint and the optimizer state and resume standard training with a possibly lower learning rate, over the new data. The number of new examples can be arbitrarily low (e.g. 1-512), and effectively acts as a replacement to in-context learning via gradient update-based few-shot learning. Due to the relatively small size of our model, this procedure also only requires a few minutes on a single GPU.

This can be seen as a form of meta-learning (Finn et al., 2017) where pretraining leads to a checkpoint which can quickly be gradient-adapted to new tasks. Note that since the language model is able to observe a "task-identifier feature", e.g. the cell name, it can perform simultaneous regression over multiple tasks simply by pretraining over all task data without any specialized techniques, and then gain the ability to regress over new tasks after fine-tuning.

3.4. Regression Scaling Paradigm

The RLM maximizes scaling on multiple axes. However, we find that the two most important scaling factors for regression are diverse training data and feature observability. Since the task of regression is fundamentally based on learning the continuity of functions, e.g. that y_1 should be close to y_2 if x_1 and x_2 are also "close", better feature observability allows the model to learn better continuous representations of inputs x, while more training data provides better coverage over the input space.

In contrast, other axes such as model size are not necessarily very important; the task of regression is inherently discriminative and does not require large models for text generation. Furthermore, sequence length requirements can especially be reduced if a user with domain knowledge can efficiently compress string representations by e.g. removing commas or whitespaces, and also placing the presumably most important features at the beginning of the string representation. For this specific paper, our default model (exact details in Appendix C) only requires a 2 layer encoder-decoder (60M parameters) with 2K maximum sequence length for sufficient results.

4. Experiments

Overall, while our results are demonstrated on performance prediction for Google's Borg compute cluster, they are meant to serve an impression of what results may appear when text-to-text regression is applied to any large system. The general conclusions are:

- Maximizing feature observability enables significantly improved regression performance, surpassing previous baselines.
- Large-scale pretraining on extensive datasets proves crucial, particularly for effective transfer learning and robust adaptation to new tasks.
- RLMs prove to be "universal" in their capabilities, allowing both pointwise prediction and density estimation, while also scaling efficiently with data, model size, and sequence length.

4.1. Data and Evaluation Protocol

Specifically for our application, we define a *task* T as a collection of 28K-56K $\{(x_i, y_i)\}_{i \geq 0}$ state-outcome pairs specifically from a cell during a month (June or November). Pairs are randomly shuffled into train/validation/test splits in a standard 8/1/1 ratio. The model may be *pretrained* on a combination of training splits from multiple tasks $\mathcal{T} = \{T_1, T_2, \ldots\}$, evaluated at test time either "in-distribution" for task T' if $T' \in \mathcal{T}$, and "out-of-distribution" if $T' \notin \mathcal{T}$. Since each task is parameterized by cell and month, an out-of-distribution evaluation can be performed across either/both unseen time and cell axes, depending on the figure (exact details in Appendix C). To maximize benchmarking quality, our paper uses a total pool of 40 cells with the largest y-value spreads.

To evaluate regression performance, we use a variety of common measurements, appropriate to the analysis. MSE is used alongside lower bound $\text{TotalVariance}_\phi$ to assess precision. To avoid bias from different y-scalings between tasks, scale-invariant Spearman rank-correlation (ρ) can also be used, especially when only rankings are sufficient, common in optimization-based applications. In ablation studies, we also use validation cross-entropy losses as simple yet accurate proxies of evaluation performance.

4.2. Results

We begin with a case study on the RLM's ability to regress from strings given enough training data, on the task T with the highest spread of y-values. In Figure 4 (left), we pretrain the RLM simultaneously

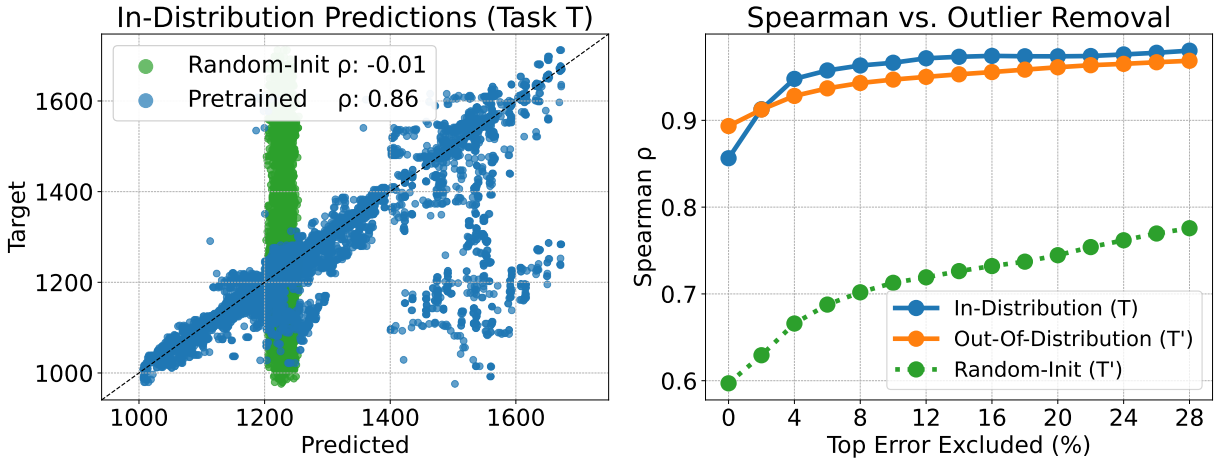

Figure 4 | **Left:** Diagonal fit (／) is better. RLM's pointwise prediction against ground truth target. **Right:** Higher is better (↑). Spearman-rank correlation of the test evaluations, after removing top outliers by MSE error.

over 8 tasks (≈1M data points, or ≈2B tokens) and evaluate in-distribution on T to achieve 0.86 rank correlation. Furthermore, when fine-tuning the pretrained checkpoint on only 512 few-shot examples from a completely new task T', we see that the RLM can also achieve an equally strong result. In contrast, a randomly initialized checkpoint only given 512 examples is unable to achieve the same level of performance in either in-distribution and out-of-distribution cases, demonstrating the importance of large-scale pretraining and transfer learning.

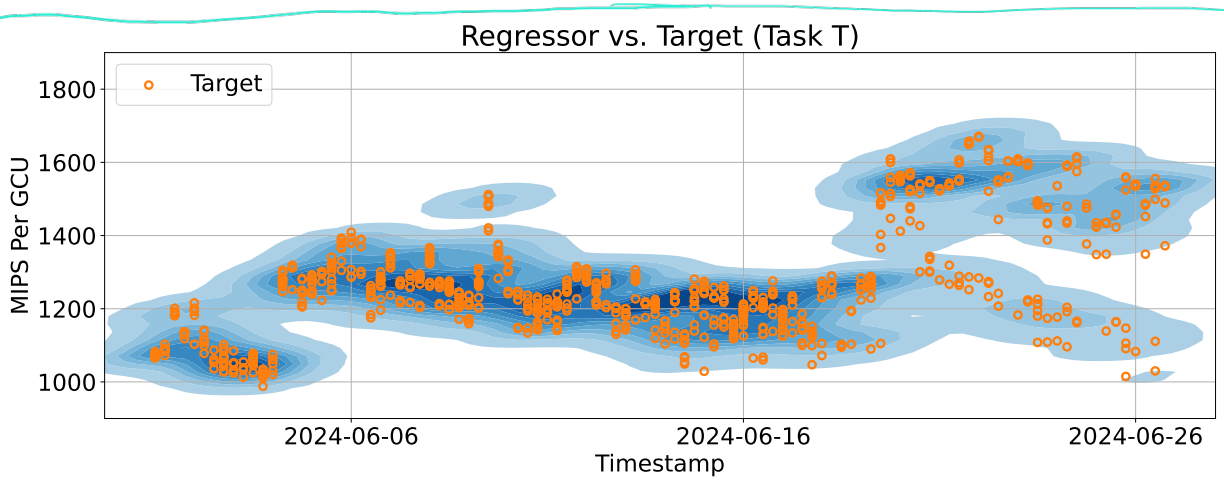

Figure 5 | Better density capture of target points is better. Kernel Density Estimate (KDE) plot of samples from $p_\theta(y|x)$ along with actual target points, over varying timestamps. Note that samples are generated from x's with distinct timestamps, while some target points may share timestamps.

Note that a large contribution to the outlier residual errors likely comes from aleatoric uncertainty, i.e. y sampled from a probability distribution $p(y|x)$ rather than a deterministic pointwise outcome. Figure 5 also visualizes the regressor's density estimation abilities when varying along a single feature in x, i.e. the time when the computation was performed. Since the RLM's representation $\phi(x)$ is very rich and unique for every state x, the training data itself does not possess any multi-modality

on y-values. However, it is remarkable that after training, the output $p_\theta(y|x)$ still expresses multiple modes, due to the RLM's inductive bias and learned continuous representations of x.

In Figure 6, we further compare the RLM's performance against theoretically optimal results achievable by baselines using limited representations $\phi(x)$, i.e. observing only tabular hyperparameter features or nothing at all ("null"), using the TotalVariance$_\phi$ lower bounds in Section 2.3. The RLM produces much lower residuals overall, with a 100x lower MSE than possible with tabular features, demonstrating the importance of maximizing feature observability.

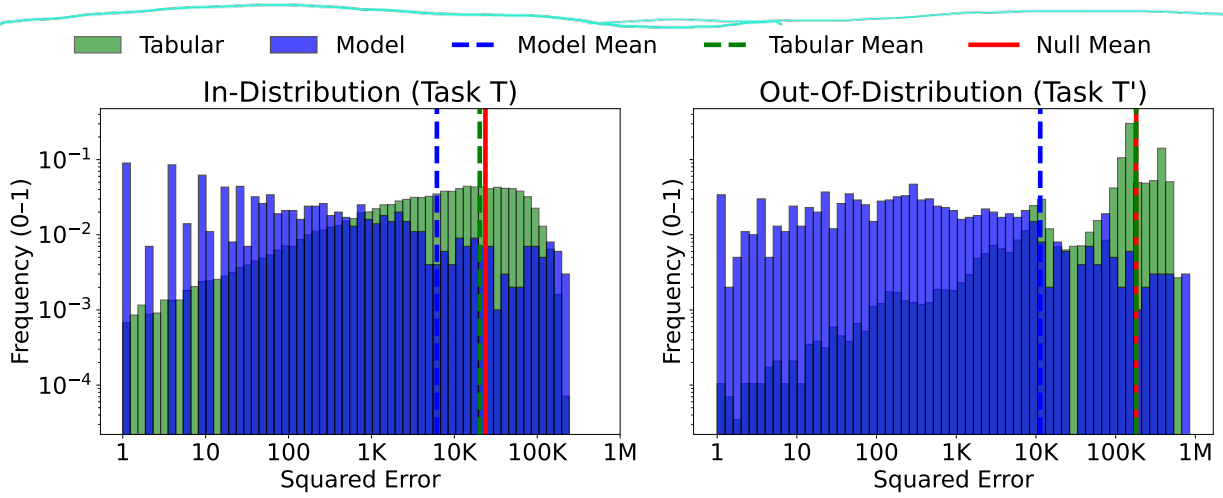

Figure 6 | Left-skewness (←) is better. Note both axes are log-scaled. **Left:** Distribution of residuals as per-sample squared error, along with mean squared error as a vertical line. **Right:** Analogous results, but for an out-of-distribution task.

In Figure 7, we demonstrate the value of large-scale pretraining, especially for transfer-learning over new tasks. Our starter checkpoints are pretrained over a set of N tasks for various N over distinct cells. While the in-distribution case shows negligible gains when varying N, the out-of-distribution case shows that a higher N leads to substantially better results on unseen cells.

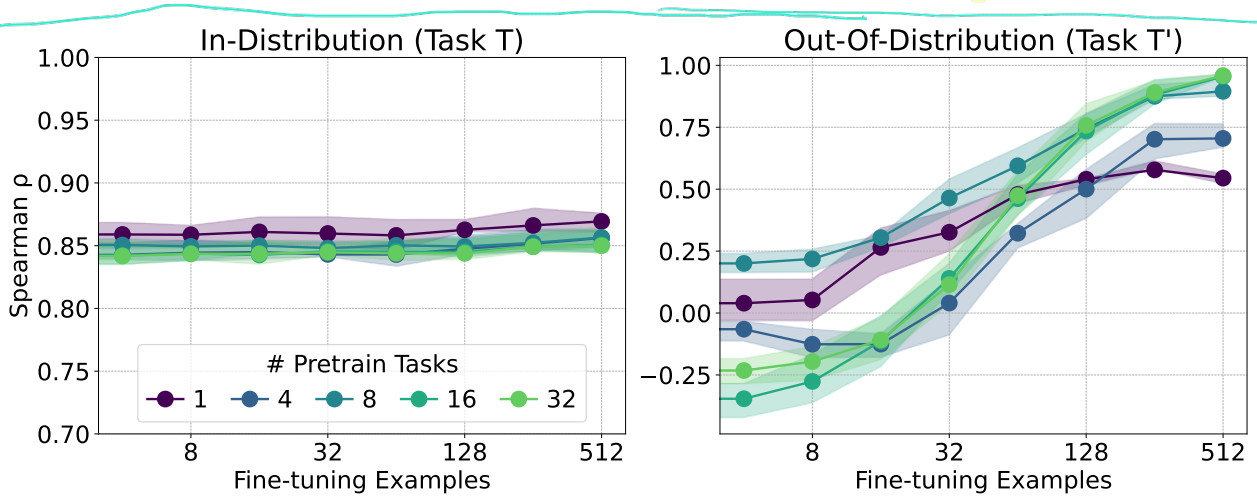

Figure 7 | Higher is better (↑). Evaluation task rank-correlation, when varying the number of finetuning examples starting from a pretrained checkpoint trained over varying numbers of tasks. Runs repeated over 10 seeds each and averaged.

Regardless if the cause of error is from inherent randomness (aleatoric) or limited data coverage (epistemic), any regression method may fundamentally be inaccurate given certain inputs. For these cases, it should at least possess a higher uncertainty (e.g. density variance), crucial to downstream applications such as Bayesian Optimization. In Figure 8, we confirm there is a high correlation between the variance of $p_\theta(y|x)$ and the residual error from a sample (x, y). Furthermore, in Figure 9, we see that the RLM's density at a single example appropriately expresses bimodality when needed, corroborating Figure 5.

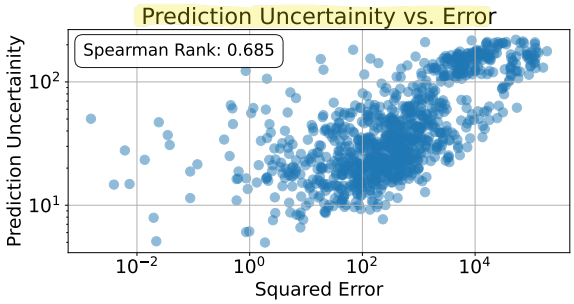

Figure 8 | Correlation between prediction uncertainty (density variance) and residual squared error.

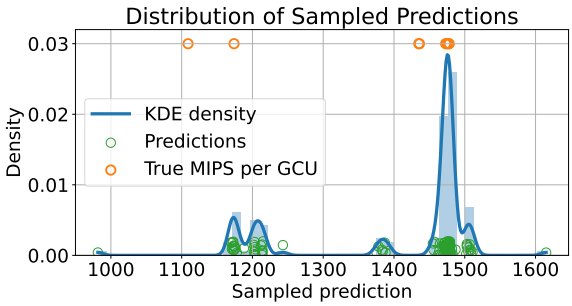

Figure 9 | RLM density $p_\theta(y|x)$ for an example input x, along with true y-values.

While the above results show the RLM's useful properties, we finally present the overall comprehensive set of benchmarking results starting from a *single* pretrained checkpoint, over various tasks denoted by C_{cell}^{month} (see Appendix B.2 for naming conventions). Multi-task results demonstrate the RLM's ability to perform inference simultaneously (when in-distribution) and via transfer learning (when out-of-distribution). Fundamentally, it is able to do so by identifying the task using the (cell, time) features and then observing other intra-task features to precisely predict the metric.

We see that the RLM can achieve very precise pointwise regression (Figure 10) on a variety of tasks with different properties and scales, when the task presumably possesses low noise. For these cases, the Spearman rank is also very high, with the majority of cases reaching above 0.93.

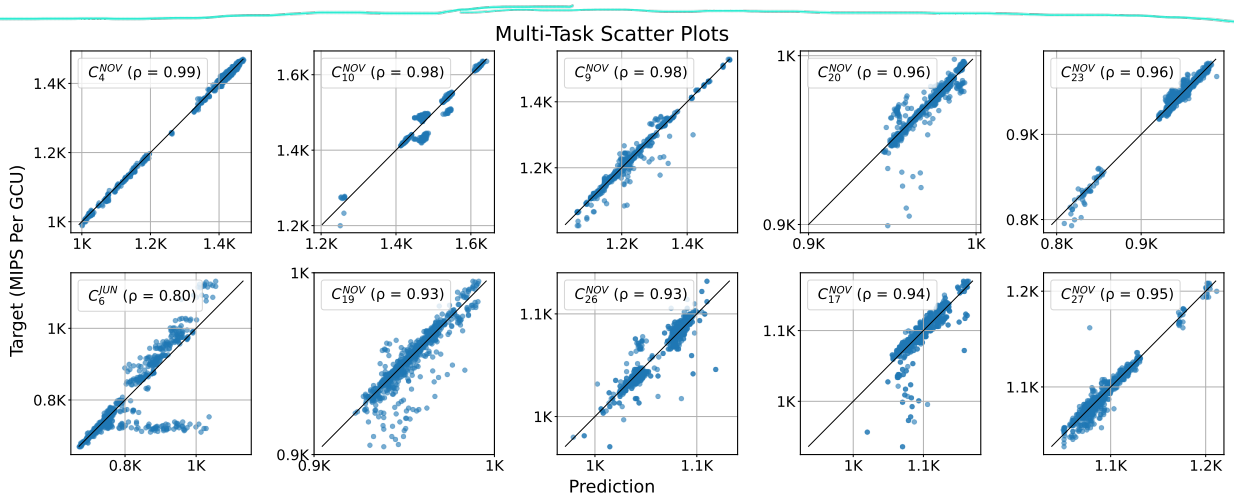

Figure 10 | Diagonal fit (／) is better. Scatter plot of predictions and ground truth targets over multiple tasks.

Furthermore, in tasks likely to possess higher aleatoric noise which make pointwise predictions inappropriate, the RLM can perform density estimations instead (Figure 11), capturing many different modes and density shapes.

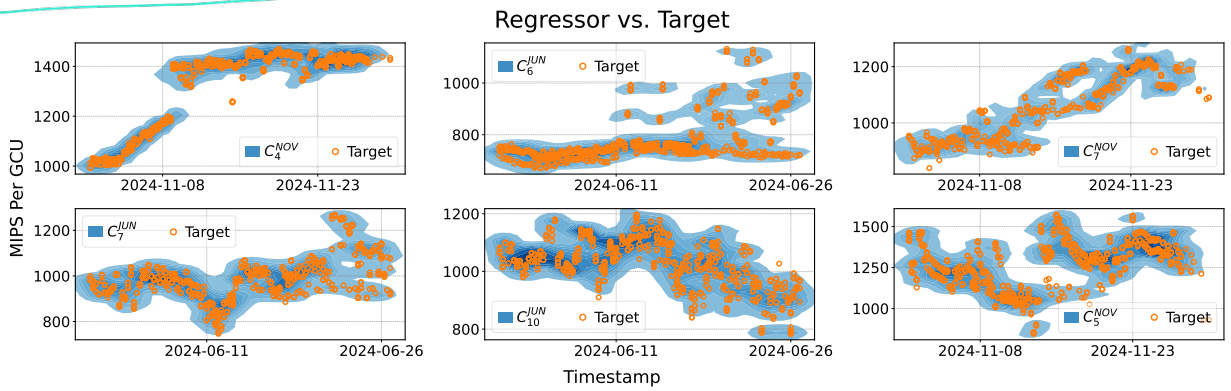

Figure 11 | Better density capture of target points is better. KDE plot over multiple tasks.

To quantify the actual gains of the RLM and to demonstrate the consistency of our technique, in Figure 12 we perform multiple runs and aggregate both the explained variance for pointwise regression and McFadden's Pseudo-R^2 (McFadden, 1974) for density estimation, respectively denoted as $R^2_{\text{EV}} = 1 - \text{MSE}_{\text{RLM}}/\text{MSE}_{\text{null}}$ and $R^2_{\text{NLL}} = 1 - \text{NLL}_{\text{RLM}}/\text{NLL}_{\text{null}}$. Explained further in Appendix A.2, these measurements capture the overall modeling gain of $p(y|x)$ from observing x against the null case $p(y)$. Interestingly, the RLM achieves near perfect prediction on task C_4^{NOV} with $R^2_{\text{EV}} \approx 1$, and even if the RLM achieves a lower R^2_{EV} on noisy tasks, it can still achieve higher R^2_{NLL} instead, as shown on tasks $C_{21}^{NOV}, C_{24}^{JUN}, C_{22}^{NOV}$.

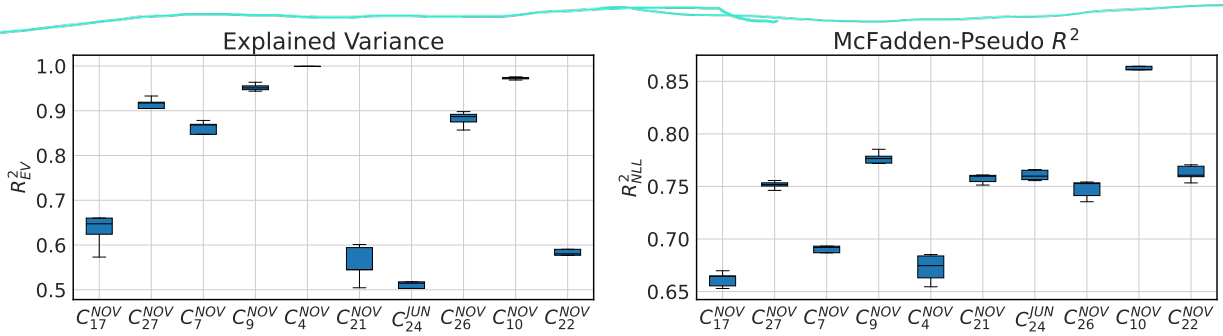

Figure 12 | Higher is better (↑). **Left:** Explained variance per task. **Right:** McFadden's Pseudo-R^2 per task. Tasks are sorted by decreasing resultant rank correlations.

5. Experiments: Ablations

5.1. Cross-Entropy Loss

We investigate the relationship between cross-entropy loss as a proxy for more traditional regression metrics such as MSE and Spearman rank correlation. During a training run, we evaluate all saved checkpoints, and based on the validation loss curve, label them as either underfitted or overfitted. In Figure 13, we see a direct correlation with MSE, while interestingly in Figure 14, we find that overfitted models can still maintain higher rank correlation.

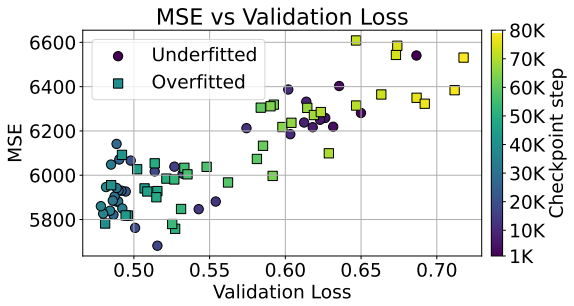

Figure 13 | Lower is better (↓). MSE across checkpoint-steps.

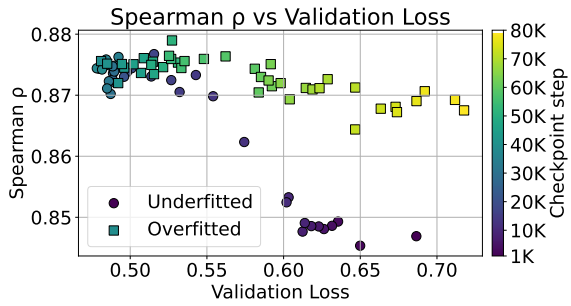

Figure 14 | Higher is better (↑). Spearman ρ across checkpoint-steps.

5.2. Architecture

In Figure 15, we show the importance of processing the input x with encoders. When accounting for equal model sizes (via layer count), we find that encoder-decoder architectures perform substantially better than decoder-only architectures even when using bidirectional attention on inputs. Note the stark contrast to LLM designs such as Gemma (Mesnard et al., 2024) and Llama (Touvron et al., 2023) which are decoder-only. We hypothesize that while decoder-only models are strong at producing outputs and chains of thought given relatively simple prompts, the information pathways routing through the decoder are insufficient to deal with complicated "prompts" x, although further investigation is needed.

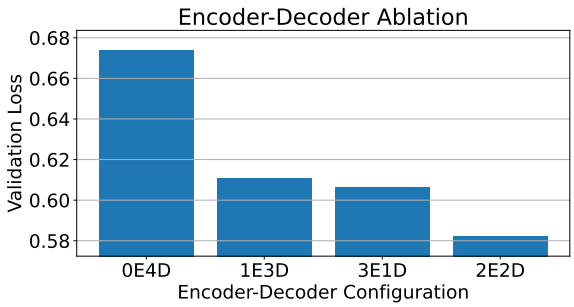

Figure 15 | Lower is better (↓). Validation losses when varying architectures. Note: "0E4D" is a decoder-only model.

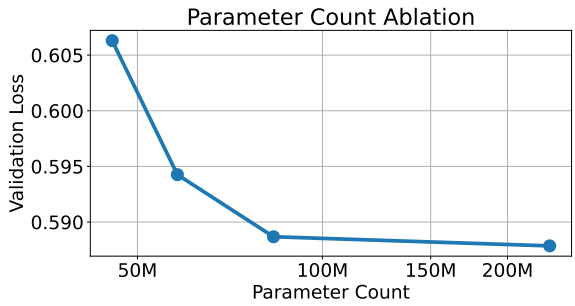

Figure 16 | Lower is better (↓). Lowest observed validation losses when varying model sizes.

In Figure 16, we also see that larger models with more parameter counts does improve regression performance, but surprisingly this quickly plateaus within the O(100M) range, which is orders of magnitudes lower than state-of-the-art general LLM models within the O(1B) range. This supports the broad utility of our text-to-text method, which requires relatively low amounts of compute, e.g. at most 1 GPU.

5.3. Feature Importance

In Figure 17, we see that increasing the input length allows the model to observe more features from x for predicting y, and thus achieving lower validation loss. When approximately $L \geq 3000$, the additional tokens mostly come from the last remaining, longest, and least important features, specifically the job-on-machine performance mentioned in Table 1, which explains the diminishing

returns with higher sequence lengths.

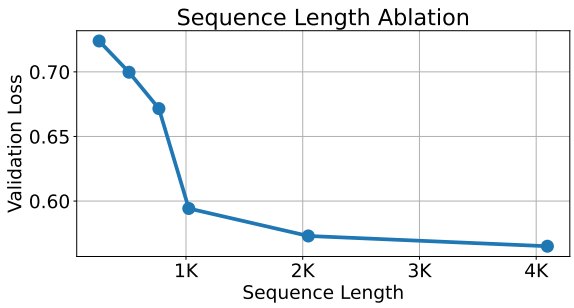

Figure 17 | Lower is better (↓). Lowest observed validation losses when training over varying maximum sequence lengths.

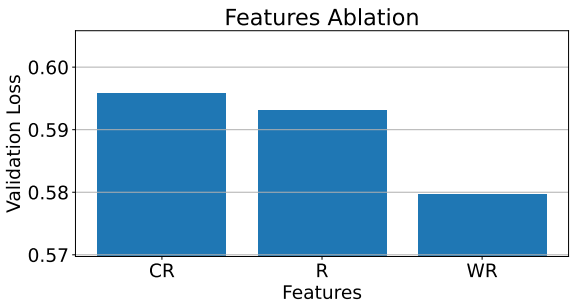

Figure 18 | Lower is better (↓). Validation losses when showing certain features ("C" = Cell, "W" = Window). "R" = using rest of the features.

In Figure 18, we also see that the model's behavior based on observing certain features aligns with our expectations from domain knowledge. For example, the performance of a cluster may heavily depend on temporal cycles – there are fewer e.g. YouTube jobs at night due to lower user counts, or certain jobs may not run over the weekend. Experimentally, the model validates this intuition, as its performance substantially improves when its x representation contains the period in which we performed bin-packing, i.e. time window feature, shown as e.g. `24-06-12T06:00:00Z` to `2024-06-12T06:05:00Z`.

5.4. Few-Shot Adaptation

Relevant for future practitioners, we provide useful ablations on the optimal settings for fine-tuning, and how they may affect results. In Figure 19, we see that finding the optimal learning rate matters significantly for reducing prediction errors when fine-tuning on multiple examples

In Figure 20, we further see that earlier checkpoints from pretraining also can lead to better results when fine-tuning on out-of-distribution tasks. This is because an overly pretrained model may "meta-overfit" to the pretraining tasks themselves, making it harder to adapt to unseen tasks, especially those more different from pretraining.

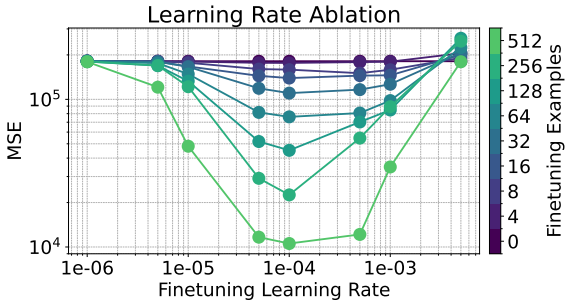

Figure 19 | Lower is better (↓). MSE after OOD fine-tuning from a fixed checkpoint, while varying the learning rate. Early checkpoint (Step 10K) is used for adaptation.

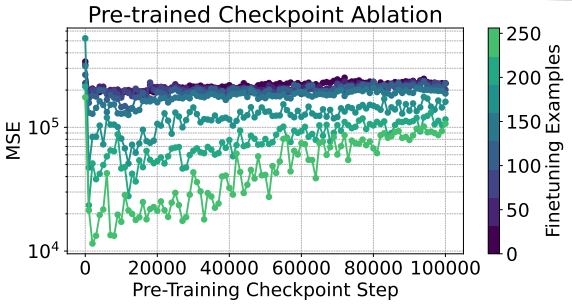

Figure 20 | Lower is better (↓). MSE after OOD fine-tuning from a specific checkpoint-step during pretraining. A fixed learning rate of 5×10^{-5} was used.

6. Conclusion

We have validated the text-to-text regression approach for performance prediction in complex, industrial environments, by specifically demonstrating its effectiveness on Google's extensive compute cluster system. Our relatively cheap and simple encoder-decoder RLM, without relying on general language pretraining, can directly train over rich, non-tabular inputs like system logs and configuration files, return highly accurate floating point predictions, and quickly adapt to new tasks with minimal extra data.

Ultimately, this work showcases RLMs as powerful, general, and scalable tools for predicting metric outcomes from raw text. They alleviate the burdens of manual feature engineering and open new avenues for creating universal simulators for complex systems. Furthermore, by accurately modeling numeric feedback from varied inputs, RLMs can serve as a foundational aspect for developing sophisticated reward models to quickly give real-world feedback and operational "experience" (Silver and Sutton, 2025), catalyzing future research on reinforcement learning for language models.

Acknowledgements

We would like to thank Uri Alon, Jonathan Lai, Mangpo Phothilimthana, Amir Yazdanbakhsh, David Smalling, Dara Bahri, Michal Lukasik, Rong-Xi Tan, Ke Xue, Shao-Hua Sun, Kuang-Huei Lee, Xinyun Chen, Chansoo Lee, Daiyi Peng, Jiyoun Ha, Aviral Kumar, Zi Wang, Gaurav Dhiman, and Yutian Chen for useful discussions and Yili Zheng, David Lo, Martin Dixon, Daniel Golovin, Denny Zhou, Claire Cui, Ed Chi, and Benoit Schillings for continuing support.

References

F. Charton. Linear algebra with transformers. *Trans. Mach. Learn. Res.*, 2022, 2022.

D. Didona, F. Quaglia, P. Romano, and E. Torre. Enhancing performance prediction robustness by combining analytical modeling and machine learning. In L. K. John, C. U. Smith, K. Sachs, and C. M. Lladó, editors, *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, January 31 - February 4, 2015*, pages 145–156. ACM, 2015. doi: 10.1145/2668930.2688047.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017.

S. Fu, S. Gupta, R. Mittal, and S. Ratnasamy. On the use of ML for blackbox system performance prediction. In J. Mickens and R. Teixeira, editors, *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, pages 763–784. USENIX Association, 2021.

D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1487–1495. ACM, 2017. doi: 10.1145/3097983.3098043.

E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. *CoRR*, abs/1910.09457, 2019.

T. Kudo and J. Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, 2018.

H. V. Madhyastha, T. E. Anderson, A. Krishnamurthy, N. Spring, and A. Venkataramani. A structural approach to latency prediction. In J. M. Almeida, V. A. F. Almeida, and P. Barford, editors, *Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference, IMC 2006, Rio de Janeriro, Brazil, October 25-27, 2006*, pages 99–104. ACM, 2006. doi: 10.1145/1177080.1177092.

D. Mahan, D. V. Phung, R. Rafailov, C. Blagden, N. Lile, L. Castricato, J.-P. Fränken, C. Finn, and A. Albalak. Generative reward models, 2024.

A. Manousis, R. A. Sharma, V. Sekar, and J. Sherry. Contention-aware performance prediction for virtualized network functions. In H. Schulzrinne and V. Misra, editors, *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020*, pages 270–282. ACM, 2020. doi: 10.1145/3387514.3405868.

D. McFadden. Conditional logit analysis of qualitative choice behavior. In P. Zarembka, editor, *Fontiers in Econometrics*, pages 105–142. Academic press, New York, 1974.

T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, P. Tafti, L. Hussenot, A. Chowdhery, A. Roberts, A. Barua, A. Botev, A. Castro-Ros, A. Slone, A. Héliou, A. Tacchetti, A. Bulanova, A. Paterson, B. Tsai, B. Shahriari, C. L. Lan, C. A. Choquette-Choo, C. Crepy, D. Cer, D. Ippolito, D. Reid, E. Buchatskaya, E. Ni, E. Noland, G. Yan, G. Tucker, G. Muraru, G. Rozhdestvenskiy, H. Michalewski, I. Tenney, I. Grishchenko, J. Austin, J. Keeling,

J. Labanowski, J. Lespiau, J. Stanway, J. Brennan, J. Chen, J. Ferret, J. Chiu, and et al. Gemma: Open models based on gemini research and technology. *CoRR*, abs/2403.08295, 2024. doi: 10.48550/ARXIV.2403.08295.

I. Schneider and T. Mattia. Carbon accounting in the cloud: a methodology for allocating emissions across data center users, 2024.

N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4596–4604. PMLR, 10–15 Jul 2018.

D. Silver and R. S. Sutton. Welcome to the era of experience, 2025.

X. Song and D. Bahri. Decoding-based regression, 2025.

X. Song, O. Li, C. Lee, B. Yang, D. Peng, S. Perel, and Y. Chen. Omnipred: Language models as universal regressors. *CoRR*, abs/2402.14547, 2024a.

X. Song, Q. Zhang, C. Lee, E. Fertig, T. Huang, L. Belenki, G. Kochanski, S. Ariafar, S. Vasudevan, S. Perel, and D. Golovin. The vizier gaussian process bandit algorithm. *CoRR*, abs/2408.11527, 2024b. doi: 10.48550/ARXIV.2408.11527.

C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, page 31–44, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936363. doi: 10.1145/1272996.1273002.

R.-X. Tan, M. Chen, K. Xue, Y. Wang, Y. Wang, F. Sheng, and C. Qian. Towards universal offline black-box optimization via learning string embedding space. In *International Conference in Machine Learning*, 2025.

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023. doi: 10.48550/ARXIV.2302.13971.

R. Vacareanu, V. Negru, V. Suciu, and M. Surdeanu. From words to numbers: Your large language model is secretly A capable regressor when given in-context examples. *CoRR*, abs/2404.07544, 2024. doi: 10.48550/ARXIV.2404.07544.

S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In K. J. Argyraki and R. Isaacs, editors, *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*, pages 363–378. USENIX Association, 2016.

A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

L. Zhang, A. Hosseini, H. Bansal, M. Kazemi, A. Kumar, and R. Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025.

S. Zhang, L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu, et al. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*, 2023.

A. Additional Theory

A.1. Epistemic Uncertainty

Following up from Section 2.3 on partial observability, when only $\phi(x)$ is observed rather than the full state x, we can quantify the gap in terms of the well known Law of Total Variance:

$$\text{Var}(y|\phi(x)) = \mathbb{E}_{x\sim p(x|\phi(x))}\left[\text{Var}(y|x)\right] + \text{Var}_{x\sim p(x|\phi(x))}\left(\mathbb{E}[y|x]\right) \tag{3}$$

Here:

- $\mathbb{E}_{x\sim p(x|\phi(x))}\left[\text{Var}(y|x)\right]$ is the average aleatoric uncertainty. It represents the expected value of the irreducible noise, averaged over all possible full states x that are consistent with the observed partial representation $\phi(x)$.
- $\text{Var}_{x\sim p(x|\phi(x))}\left(\mathbb{E}[y|x]\right)$ is the crucial term representing the contribution of epistemic uncertainty to the total variance. It is the variance of the true conditional mean $\mathbb{E}[y|x]$ (the optimal prediction if x were known) due to the unobservability of x given only $\phi(x)$.

Since $\text{Var}_{x\sim p(x|\phi(x))}\left(\mathbb{E}[y|x]\right) \geq 0$, it follows that

$$\text{Var}(y|\phi(x)) \geq \mathbb{E}_{x\sim p(x|\phi(x))}\left[\text{Var}(y|x)\right] \tag{4}$$

This inequality formally shows that observing only a partial representation $\phi(x)$ increases (or at best, keeps the same) the variance that lower-bounds the regressor's achievable error, compared to the average aleatoric variance. The optimal performance of a regressor observing $\phi(x)$ is now limited by $\text{Var}(y|\phi(x))$. Thus, the limited observability captured by $\phi(x)$ induces epistemic uncertainty which directly contributes to a higher overall variance, further limiting the regressor's optimal performance.

A.2. Explained Negative Log-Likelihood

It is well known that the traditional explained variance R^2_{EV} can be computed in terms of MSE ratios, i.e. $1 - \text{MSE}_{\text{RLM}}/\text{MSE}_{\text{null}}$, where the null case simply corresponds to the variance of the total y-population.

McFadden (1974) also provides an analogous expression for density estimation cases, where negative log-likelihood (NLL) is instead used to calculate performance:

$$R^2_{\text{NLL}} = 1 - \text{NLL}_{\text{RLM}}/\text{NLL}_{\text{null}} \tag{5}$$

Note that simply using absolute NLL_{RLM} as the final measurement is ambiguous, since there may not be an established reference for what is "good" or "bad". Instead, the reference is chosen as NLL_{null} corresponds to a density estimator which models y *without* using x. Here, we choose the RLM itself, due to its decoder's universal density approximation abilities (Song and Bahri, 2025) over more constrained distributions such as Gaussians. Note that NLL_{null} is also an empirical approximation of the true entropy, which is impossible to calculate precisely as it requires online access to the true distribution.

B. Additional Data Information

B.1. Example String Representation

In Figure 21, we show what the model exactly observes. Note the natural use of the newline character "\n" as a natural separator between different features.


```
'\ncell: cell_a\n\n\'2024-06-12T06:00:00Z\'\n\n\'2024-06-12T06:05:00Z\'\n\nLarge users: [\'datapipeline-prod\', \'researcher_human_1\',
 \'monitoring-jobs\', \'model-server-public-prod-jobs\', \'ecommerce\', \'ads_fetch\', \'researcher_human_2\', \'real-time-diagnostics
\', \'storage_1\', \'storage_2\', \'storage_3\', \'webserver\', \'storage_aggregation\', \'storage_frontend\', \'researcher_human_5\',
 \'routing\', \'storage_4\', \'researcher_human_4\', \'storage_5\', \'analytics-storage_6\']\nsearch_space: {\'machineA\': [\'machineA
\', \'none_selected\']} {\'machineD,machineC\': [\'machineD\', \'machineC\', \'none_selected\']} {\'machineA,machineE\': [\'machineA\',
 \'machineE\', \'none_selected\']} {\'machineF\': [\'machineF\', \'none_selected\']} {\'machineD,machineA\': [\'machineD\', \'machineA
\', \'none_selected\']} {\'machineD\': [\'machineD\', \'none_selected\']} {\'machineE\': [\'machineE\', \'none_selected\']} {\'
machineD,machineE\': [\'machineD\', \'machineE\', \'none_selected\']} {\'JOB/storage_4/PRODUCTION_WORKLOAD\': [\'machineE\', \'
none_selected\']} {\'JOB/processing_1/me.processing_1.processing_1/PRODUCTION_WORKLOAD\': [\'machineE\', \'none_selected\']}\n\
nassignments: {"JOB/storage_4/PRODUCTION_WORKLOAD": "none_selected", "JOB/processing_1/me.processing_1.processing_1/
PRODUCTION_WORKLOAD": "machineE", "machineD": "machineD", "machineD,machineA": "machineA", "machineD,machineE": "machineD", "machineD,
machineC": "none_selected", "machineA": "machineA", "machineA,machineE": "machineE", "machineF": "machineF", "machineE": "
none_selected"}\ncell: cell_a\ndistributions:\n- platform: {machineA}\n num_machines: 1.239e+03\n low_level_zones: 5.200e+01\n
mid_level_zones: 5.200e+01\n high_level_zones: 4.300e+01\n resources: 5.481e+05\n- platform: {machineE}\n num_machines: 1.070e+02\n
low_level_zones: 3.900e+01\n mid_level_zones: 3.900e+01\n high_level_zones: 3.500e+01\n resources: 2.430e+04\n- platform: {machineD}\n
 num_machines: 5.640e+02\n low_level_zones: 3.000e+01\n mid_level_zones: 3.000e+01\n high_level_zones: 1.900e+01\n resources: 2.769e
+05\n- platform: {machineC}\n num_machines: 6.100e+01\n low_level_zones: 6.000e+00\n mid_level_zones: 6.000e+00\n high_level_zones:
3.000e+00\n resources: 2.306e+04\n- platform: {machineF}\n num_machines: 1.880e+02\n low_level_zones: 1.200e+01\n mid_level_zones:
1.200e+01\n high_level_zones: 6.000e+00\n resources: 4.527e+04\n\njob_profiles:\n- job: {user: leaf, group_name: bun}\n
platform_profiles:\n machineD: {mean_productivity_per_resource_usage: 1.048e+00, mean_mips_per_resource_usage: 8.588e+02}\n machineF:
{mean_productivity_per_resource_usage: 1.060e+00, mean_mips_per_resource_usage: 8.285e+02}\n machineE: {
mean_productivity_per_resource_usage: 1.100e+00, mean_mips_per_resource_usage: 8.292e+02}\n machineC: {
mean_productivity_per_resource_usage: 9.721e-01, mean_mips_per_resource_usage: 7.911e+02}\n observed_resource_limit: 7.861e+02\n
observed_num_vms: 9.000e+00\n limits: {machine_limit: 1.000e+00, switch_limit: 1.000e+00, low_level_limit: 1.000e+00, has_port_limit:
true, job_requested_resource_limit: 8.832e+02, job_requested_num_vms: 10, group_requested_resource_limit: 8.832e+02}\n
productivity_per_resource_1: 1.048e+00\n productivity_per_resource_2: 1.045e+00\n- job: {user: cent, group_name: rock}\n
platform_profiles:\n machineD: {mean_productivity_per_resource_usage: 3.803e+01, mean_mips_per_resource_usage: 5.716e+02}\n machineA:
{mean_productivity_per_resource_usage: 7.685e+00, mean_mips_per_resource_usage: 4.902e+02}\n machineE: {
mean_productivity_per_resource_usage: 5.893e+01, mean_mips_per_resource_usage: 5.610e+02}\n machineC: {
mean_productivity_per_resource_usage: 3.274e+01, mean_mips_per_resource_usage: 7.173e+02}\n observed_resource_limit: 6.400e+02\n
observed_num_vms: 2.000e+01\n limits: {job_requested_resource_limit: 6.400e+02, job_requested_num_vms: 20,
group_requested_resource_limit: 6.400e+02}\n productivity_per_resource_1: 1.941e+01\n productivity_per_resource_2: 3.435e+01\n- job: {
user: pond, group_name: green}\n platform_profiles:\n machineD: {mean_productivity_per_resource_usage: 1.413e+06,
mean_mips_per_resource_usage: 7.420e+02}\n observed_resource_limit: 1.100e+02\n observed_num_vms: 1.900e+01\n limits: {
job_requested_resource_limit: 1.362e+04, job_requested_num_vms: 1401, group_requested_resource_limit: 1.362e+04}\n
productivity_per_resource_1: 1.413e+06\n productivity_per_resource_2: 1.413e+06\n
```


Figure 21 | Example anonymized string representation of x, truncated to fit.

B.2. Data Organization and Statistics

Due to corporate privacy concerns, we are required to anonymize all cell names, but make sure the naming scheme is consistent across all figures. Each task is labeled as C_{cell}^{month}, parameterized by cell (integer index) and month (name). The index of a cell is sorted in reverse order according to its "spread" of y-values, i.e. variance of $p(y)$ unconditioned on x, and we show the distribution of spreads in Figure 22. Thus C_1^{JUN} and C_2^{JUN} within June and C_1^{NOV} and C_2^{NOV} within November are the highest-spread cells, with June cells having dataset sizes 54K-56K and November having 28K.

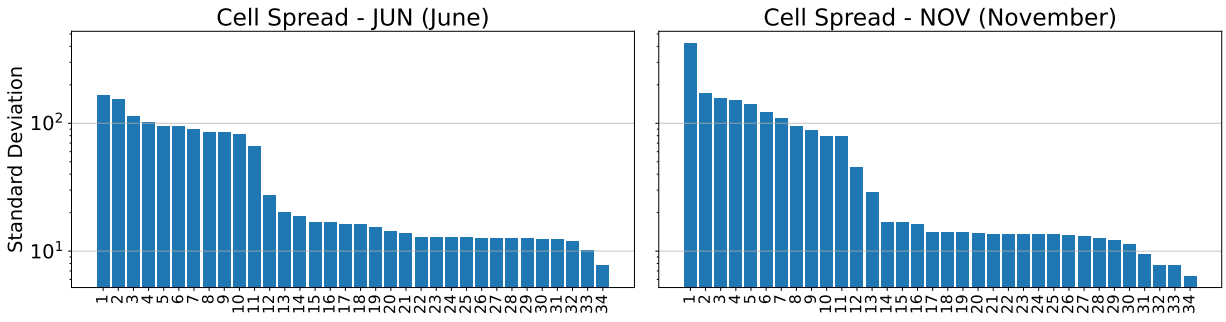

Figure 22 | Spread of cells included in our study, separated by month.

C. Experimental Settings

C.1. Default Settings

We used the standard T5X EncoderDecoder which can found in the open-source codebase https://github.com/google-research/t5x. Below are default settings (unless otherwise overridden).

For training/pretraining:

- Optimizer: Adafactor (Shazeer and Stern, 2018) with base learning rate 0.1 and square root decay with 1000 warm-up steps, 0.5 decay factor, 2000 steps per decay, and 10000 steps per cycle. Batch size 128 with 8 microbatches.
- Vocabulary and Tokenizer: SentencePiece tokenizer (Kudo and Richardson, 2018) with T5X's default vocabulary of 32000 subword tokens, in addition to the custom P10 tokens (Charton, 2022) for representing y-objectives, with 4-digit mantissas.
- Early stopping: We train for a maximum of 100K steps, but early stop based on validation loss if overfitting is detected.
- Architecture: 2 encoder layers, 2 decoder layers, 16 heads, 64 head dimension, 512 embedding dimension, 2048 MLP dimension, leading to 58M parameters. Default sequence length 2048, with truncation occurring afterwards.

For fine-tuning:

- Optimizer: Restore checkpoint optimizer state, but change the learning rate to 5×10^{-5}.
- Batch size: 128. Note that if the finetuning data size is lower, we simply repeat up to this size.
- Early stopping: Perform up to 200 epochs of finetuning, but early stop based on validation loss if overfitting is detected.

For inference:

- Sampling: 128 parallel decoding examples per single x, while removing accidental "extreme" outliers outside of the known range [500, 3000] of possible achievable MIPs per GCU values.
- Pointwise Aggregation: Take the numeric mean of samples (to minimize MSE) or numeric median (to minimize Spearman rank).

C.2. Main Training Runs

All of the figures in this paper stem from a few main training setups, focused on evaluating on both in-distribution and out-of-distribution cases, or ablations. Specifically, these main settings:

- Limit Testing: We train a larger model (267M params) with 4096 sequence length and batch size 256, over nearly all tasks except the highest spread cells across the two months $\{C_1^{JUN}, C_1^{NOV}\}$. We then also evaluate on the highest spread cells possible across the two months, i.e. C_2^{JUN} (in-distribution) and C_1^{NOV}.
- Adaptation Testing: We train five separate checkpoints over $\{1, 4, 8, 16, 32\}$ different tasks chosen randomly, but making sure there is a uniquely seen task C_2^{JUN} by all checkpoints, and also evaluate on a out-of-distribution task C_{10}^{NOV} unseen by all checkpoints.
- Ablations: We simply pretrain on 7 high spread tasks from June.

More specifically, the pretraining tasks for Limit Testing are the combination of C_2^{JUN} to C_{10}^{JUN}, C_{22}^{JUN} to C_{31}^{JUN}, C_2^{NOV} to C_{10}^{NOV}, C_{17}^{NOV}, and C_{19}^{NOV} to C_{27}^{NOV}.

For Adaptation Testing, the pretraining tasks were:

- 1 Cell: C_2^{JUN}
- 4 Cells: $C_2^{JUN}, C_9^{JUN}, C_{12}^{NOV}, C_{30}^{JUN}$
- 8 Cells: $C_2^{JUN}, C_3^{JUN}, C_{10}^{JUN}, C_{25}^{JUN}, C_{31}^{JUN}, C_2^{NOV}, C_{28}^{NOV}, C_{30}^{NOV}$
- 16 Cells: $C_2^{JUN}, C_6^{JUN}, C_9^{JUN}, C_{25}^{JUN}, C_{27}^{JUN}, C_{28}^{JUN}, C_{29}^{JUN}, C_{31}^{JUN}, C_2^{NOV}, C_5^{NOV}, C_8^{NOV}, C_{12}^{NOV}, C_{30}^{NOV}, C_{32}^{NOV}, C_{33}^{NOV}, C_{34}^{NOV}$
- 32 Cells: $C_1^{JUN}, C_2^{JUN}, C_3^{JUN}, C_4^{JUN}, C_6^{JUN}, C_7^{JUN}, C_8^{JUN}, C_9^{JUN}, C_{10}^{JUN}, C_{23}^{JUN}, C_{24}^{JUN}, C_{25}^{JUN}, C_{26}^{JUN}, C_{27}^{JUN}, C_{28}^{JUN}, C_{29}^{JUN}, C_{30}^{JUN}, C_{31}^{JUN}, C_1^{NOV}, C_2^{NOV}, C_5^{NOV}, C_{11}^{NOV}, C_{12}^{NOV}, C_{13}^{NOV}, C_{14}^{NOV}, C_{16}^{NOV}, C_{18}^{NOV}, C_{30}^{NOV}, C_{31}^{NOV}, C_{32}^{NOV}, C_{33}^{NOV}, C_{34}^{NOV}$

For Ablations, the pretraining tasks were C_1^{JUN} to C_8^{JUN}.

C.3. Individual Figures

Distribution Of Input String Length, Figure 3: For this, we use the evaluation results from Limit Testing and simply make the histogram of the inputs used across all cells during test-time.

Outlier Sensitivity In Rank Correlation, Figure 4: We used the 8-cell pretrained model from Adaptation Testing, and also use the same cells for evaluation. Note that all cells are technically "out-of-distribution" for a randomly initialized checkpoint.

We use the same fine-tuning procedure for all model tests. We repeat the fine-tuning procedure 10 times and report the mean for the Spearman rank correlation. We pick a single seed for our Diagonal Fit scatter-plot.

Density Capture by RLM, Figure 5: From Adaptation Testing, we use the 8-cell out-of-distribution result. We take the predictions as well as samples (128 samples per input), and make a density plot of the metric (MIPS per GCU).

MSE Histogram for In and Out-Of-Distribution Adaptation, Figure 6: We use the Limit Testing setting, with pretrained model checkpoint at 15K steps, which achieved the lowest validation loss within 5K step intervals. Recall from Section 2.3 that to obtain the theoretically optimal tabular and null residuals, we grouped all y's into the equivalence classes and took the mean y-value of each equivalence class as the theoretically optimal prediction.

Pretraining Diversity and Few-Shot Adaptation, Figure 7: From Adaptation Testing, we plot all of the results. For fine-tuning experiments, we use pretraining checkpoint early stopped at 10K steps, motivated by our checkpoint-learning rate analysis in Figure 20, and observing validation loss.

Prediction Uncertainty and Multi-modality, Figure 8 and Figure 9: We simply fine-tune the 8-cell pretrained base model from Adaptation Testing on in-distribution task C_2^{JUN} for 512 samples, and pick a seed.

For Figure 8, we report the scatter plot of prediction standard deviation across its 128 generated samples per input on the Y axis as prediction uncertainty, and squared-error on the X axis.

For Figure 9, we identify a set of time-stamps where the ground truth target demonstrates bi-modal distribution, and pick one of these time-stamps. At this exact time-stamp, we have one input and 128 generated regressor predictions. We plot the ground-truth values observed at that time-stamp, as well as the 128 generated values (Sampled Predictions). We then plot the histogram and density plot on the sampled predictions.

Scatter and Density Plots, Figure 10 and Figure 11: We use the Limit Testing setting, with pretrained model checkpoint at 15K steps. We fine-tune the model for each task individually, over 512 examples, with a learning rate of 1×10^{-4} for a single seed.

Explained Variance and Negative Log-Likelihood, Figure 12: We use the Limit Testing setting, with pretrained model checkpoint 15K steps. To reliably estimate NLL_{null}, we fine-tune the randomly initialized variant of the Limit Testing model on *empty strings* with 1024 examples. NLL_{RLM} is also fine-tuned on 1024 examples. Learning rate is 1×10^{-4}, and we report results over 5 seeds.

Validation Loss and MSE, Spearman ρ, Figure 13 and Figure 14: We train a new base model, with 4 layers, 4096 sequence length, on high spread cells (C_2^{JUN}, C_3^{JUN}, C_5^{JUN}, C_7^{JUN}, C_8^{JUN}, C_9^{JUN}, C_{10}^{JUN}) and save checkpoints at 1K step intervals. We then evaluate every checkpoint on in-distribution cell C_2^{JUN} over 100 seeds, without any fine-tuning. We report the mean MSE and Spearman-ρ over 100 seeds.

Encoder-Decoder Ablation, Figure 15: We train four models on the Ablation setting with the same default model dimensions, but with different encoder and decoder layers. `(0E4D, 1E3D, 3E1D, 2E2D)` had roughly the same number of parameters `(62.3M, 60.2M, 58M, 56M)` respectively, and minimum validation loss was observed at step `(18K, 17K, 17K, 23K)` respectively.

Parameter Size Ablation, Figure 16: We train four models, of sizes `(45.5M, 58.1M, 83.2M, 234.3M)` (2 layers, 4 layers, 8 layers and 32 layers respectively), with default model dimensions on the Ablation setting, but with a changed sequence length of 1024. We then measure the minimum validation loss, achieved at respective steps (27K, 16K, 13K, 11K).

Sequence-Length Ablation, Figure 17: We train six models at sequence lengths of `(4096, 2048, 1024, 768, 512, 256)`, with default model size and dimensions on the Ablation setting. We measure the minimum validation loss, achieved at respective steps `(26K, 23K, 16K, 18K, 13K, 12K)`.

Feature Ablation, Figure 18: We train three models all of default dimensions but with a changed sequence length of 1024, and measure the minimum validation loss, achieved at step `(22K, 20K, 23K)` for `(CR, R, WR)` respectively on the Ablation setting.

Checkpoint and Learning Rate Ablation, Figure 19 and Figure 20: We use the default model size but with a sequence length of 4096. This model is trained on C_2^{JUN} to C_{10}^{JUN} and then fine-tuned and evaluated on the highest-spread out-of-distribution task C_1^{NOV}.

For Figure 19, we pick the checkpoint at 10K steps and fine-tune the base model on C_1^{NOV} at learning rates of $(5 \times 10^{-3}, 1 \times 10^{-3}, 5 \times 10^{-4}, 1 \times 10^{-4}, 5 \times 10^{-5}, 1 \times 10^{-5}, 5 \times 10^{-6}, 1 \times 10^{-6})$ for `(0, 4, 8, 16, 32, 64, 128, 256)` examples each. We report the mean of MSE over 10 fine-tuning seeds.

For the checkpoint ablation in Figure 20, we set a fine-tune learning rate of 5×10^{-5} and take checkpoints at every-step, starting from 1K up to 100K. We fine-tune every checkpoint of the base model on C_1^{NOV} with validation-loss based early stopping, for `(0, 4, 8, 16, 32, 64, 128, 256)` fine-tuning examples. Due to the scale of the study, we only use one seed.