

UNDERSTANDING INDUCTIVE BIAS IN THE ERA OF
LARGE-SCALE PRETRAINING WITH SCIENTIFIC DATA

by

Nathaniel Gruver

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE
NEW YORK UNIVERSITY
MAY, 2025

Professor Andrew Gordon Wilson

© NATHANIEL GRUVER
ALL RIGHTS RESERVED, 2025

Yes, there were times, I'm sure you knew
When I bit off more than I could chew
But through it all, when there was doubt
I ate it up and spit it out
I faced it all, and I stood tall
And did it my way

—Frank Sinatra

DEDICATION

For my family. You taught me to love learning, and you were always there when I needed you.

ACKNOWLEDGEMENTS

First, to Andrew, thank you for seeing the potential in me and for pushing me to be a better researcher. I benefited from your drive to work on fundamental topics and your passion for clear writing. You maintain a strong culture of collaboration in your lab, and it was a consistent source of support and inspiration.

To Kyunghyun, thank you for always lending your ear and wisdom in difficult moments. You are a special blend of magnanimous and pragmatic. Our meetings made me feel energized about research, especially when they touched on non-autoregressive sequence models, which you championed before they were fashionable.

Among my peers, I probably spent the most time with, and subsequently learned the most from, Marc Finzi, Sam Stanton, and Alan Amin. Sam and Marc helped me get unstuck early in PhD and helped me gain confidence. It would have looked different without them. Marc, thank you for countless fun conversations and brainstorming sessions. You have an infectious curiosity about almost everything, and you are a kind and generous person. Sam, the nights we spent manically messaging over slack were some of the most fun in the PhD. You're a deep thinker, a straight-shooter, and a reliable friend. Lastly, to Alan, I'm so grateful you joined the group for the last two years of my PhD. It wouldn't have been the same without all the laughs we shared in your bathroom-adjacent office. You're laudable for your research talent but also just for being fun.

During my time at NYU, I also worked with many other great collaborators, includ-

ing Greg Benton, Wesley Maddox, Shikai Qiu, Polina Kirichenko, Pavel Izmailov, Sanyam Kapoor, Micah Goldblum, Lily Li, and Yilun Kuang. Greg, thanks for being my officemate in my third year. You made that year much more fun and you imparted real wisdom on several occasions. Wesley, thank you for bringing consistent contributions and upbeat energy to our projects. Shikai, it would have been fun to work together more. The summer we spent chatting over dinner at FAIR was one of my favorites during the PhD. Polina, thank you for offering empathy during difficult moments in my first year in the PhD. Pavel, thank you for all the pick-up basketball and sage advice. Sanyam, sharing a laugh with you throughout the PhD was always a relief, especially when we were navigating a rocky start during COVID. Micah, thank you for always speaking your mind in the group and for your encouragement. Lily and Yilun, thanks for your consistent hard work and for keeping me company in the office the last two years. To Sanae Lotfi, Andres Potapczynski, Fred Lu, and Ethan Baron, I'm sad we didn't end up writing a paper together, but I'm grateful for a few heart-to-hearts we shared and I'm glad we crossed paths.

During internships and collaborations with FAIR, Prescient Design and Together, I was also lucky to collaborate with Nathan Frey, Zack Ulissi, Larry Zitnick, Ben Athiwaratkun, and Alex Rives. All of you were extremely encouraging of me and my career at one time or another and it left a positive mark. All of you are talented researchers but also good people.

Outside of research, I wouldn't have made it through without the support of my friends and family. Mom and Dad, thank you for always picking up the phone and for always giving me the tough love I needed. Alex, thank you for loving my demon mode but also for encouraging my better angels and reminding me to follow my heart. Ben, Avery, and Sabar, the first two years we lived together were such a joyful introduction to New York. That was a unique environment driven by curiosity and adventure. Ben, thank you for putting up with me when we first moved, for countless long conversations, and for being a good influence in myriad ways. Avery, I'm grateful for our many heart-to-hearts, whether at night in the

New York apartments, in a Kayak in Florida, or over barbecue in Osaka. You're a unique mix of chaotic and lawful energy and a wellspring of wisdom in your best moments. Sabar, our shared studio space was a refuge for me in my second and third year. Thank you for jumping headlong into that adventure and many others we've shared over the years.

I'm also grateful for my Boston friends (Jack, Brian, Ilya, Pete, and Ian), who kept me company my first summer back on the East Coast and on many trips since, Stanford friends that kept in touch (Derek W., Mike, Allan, Andrew N., Andrew B., Alvin, Alka, Sarah, Sherman), and recent friends from MIT (Hannah, Nikos, Derek L., Sharut, and Victor) who let me tag along in Budapest, Vienna, and Vancouver. Over the years, New York itself was also a great source of friends, including Annie, Leo, and Nisa. Lastly, I want to thank a few musicians who formed the theme track for my PhD: Donny Hathaway, Erykah Badu, Jordan Rakei, Bonnie Raitt, Lianne La Havas, Aphex Twin, Jon Hopkins, Q-Tip, Kendrick Lamar, Little Simz, J Dilla, Caroline Polachek, Geese, and, of course, Joni Mitchell.

ABSTRACT

Inductive biases are crucial for machine learning in data-scarce settings, but their optimal role in data-rich regimes remains poorly understood. This thesis challenges the conventional wisdom that strict architectural constraints are necessary for modeling numerical data, particularly in physics and chemistry. Through systematic empirical studies, I demonstrate that data-driven approaches can effectively learn both physical symmetries and broader numerical patterns without explicit architectural constraints. First, I show that transformer models trained with data augmentation can acquire stronger equivariance properties than convolutional neural networks, despite lacking built-in symmetry constraints. Building on this insight, I investigate whether pretrained language models can learn generalizable numerical capabilities from text alone. By studying the behavior of language models in many settings, I demonstrate that text pretraining induces a preference for simple functions that serves as a powerful inductive bias across numerical domains. This emergent bias enables large language models to outperform specialized architectures on benchmark tasks in time series forecasting and 3D structure prediction, achieving state-of-the-art results with minimal task-specific adaptation. However, these benefits do not extend universally - I identify molecular property prediction as a key limitation and trace this failure to fundamental constraints in discrete token representations. This work provides a comprehensive framework for understanding when learned biases can replace architectural constraints in numerical domains, with important implications for model design in scientific machine learning.

Contents

Dedication	iv
Acknowledgments	v
Abstract	viii
List of Figures	xvi
List of Tables	xxxiii
List of Appendices	xxxv
1 Introduction	1
I Distilling Architecture Constraints for Learning Symmetric Functions	4
2 Deconstructing the Inductive Biases of Hamiltonian Neural Networks	6
2.1 Introduction	6
2.2 Related Work	8
2.3 Background	9
2.4 Deconstructing Hamiltonian neural networks	10
2.4.1 Energy conservation	10

2.4.2	Symplectic vector fields	12
2.4.3	Second-order structure	14
2.4.4	Functional complexity	16
2.5	Distilling the minimal inductive biases	17
2.6	Conclusion	20
3	The Lie Derivative for Measuring Learned Equivariance	21
3.1	Introduction	21
3.2	Background	22
3.3	Image classification and translational equivariance	23
3.3.1	Continuous signals and aliasing	23
3.4	Related Work	25
3.5	The Lie derivative	26
3.5.1	Mathematical formulation	26
3.5.2	Efficient implementation	28
3.5.3	Alternative metrics	29
3.6	Layer-wise effects on equivariance	29
3.7	Trends in learned equivariance	32
3.7.1	Methodology	32
3.7.2	Equivariance across architectures	33
3.7.3	Effects of Training and Scale	34
3.7.4	Equivariance out of distribution	35
3.7.5	Why aren't CNNs more equivariant than ViTs?	36
3.7.6	Learning rotation equivariance	36
3.8	Conclusion	37

II	Sequence Modeling for Numerical Data: Transfer Success and Limitations	40
4	Large Language Models Are Zero-Shot Time Series Forecasters	42
4.1	Introduction	42
4.2	Background	44
4.3	Related work	46
4.4	Continuous densities with autoregressive models	46
4.5	LLMTime: retrofitting LLMs for forecasting	47
4.6	Time series evaluations	49
4.7	LLMTime evaluation	51
4.7.1	Deterministic forecasts	53
4.7.2	Probabilistic forecasts	54
4.7.3	Temporal holdouts	55
4.8	Connections with simplicity bias	57
4.9	Promises of jointly modeling text and numerical data	59
4.10	Heterogeneous scientific data	63
4.11	Conclusion	64
5	Fine-Tuned Large Language Models Generate Stable Inorganic Materials as Text	66
5.1	Introduction	66
5.2	Related work	67
5.3	Background	70
5.4	Parameterizing bulk materials	71
5.5	Fine-tuning approach	71
5.5.1	Dataset and Augmentations	71
5.5.2	String encoding and prompts	73

5.6	Evaluating samples	75
5.6.1	Energy above hull	75
5.7	Results	77
5.7.1	Unconditional generation	77
5.7.2	Conditional sampling	79
5.8	Connections with simplicity bias and LLMTime	81
5.9	Conclusion	83
6	Open Challenges in Applying Language Models to Numerical Data	85
6.1	Introduction	85
6.2	Related Work	87
6.3	Preliminaries	88
6.3.1	Motivating problem setting	88
6.3.2	Computational chemistry as a test bed	89
6.4	Theoretical limitations	90
6.5	Practical challenges	91
6.6	Experimental Setup	92
6.7	Model Architecture	95
6.8	Tokenization	100
6.9	Text Pretraining	102
6.10	Conclusion	104
7	Conclusion	106
A	Appendix: Deconstructing Learned Symmetries	109
A.1	Appendix Outline	109
A.2	Mathematical Details	109

A.2.1	Energy Conservation for Neural ODEs	109
A.2.2	HNN Energy Conservation	111
A.2.3	Symplecticity	112
A.3	Mujoco Experiment Details	112
A.4	Additional Experimental Results	114
A.4.1	Comparison of loss functions	114
A.4.2	Additional Systems	114
A.5	Aliasing Extended Discussion	115
A.6	Lie Groups, Lie Derivatives, and LEE	117
A.6.1	Lie Groups and Local/Global Notions of Equivariance	117
A.6.2	Lie Derivative Chain Rule	118
A.6.3	Stochastic Trace Estimator for Layerwise Metric	118
A.7	LEE Theorems	119
A.7.1	LEE and consistency regularization	119
A.7.2	Translation LEE and aliasing	121
A.8	Learned Equivariance Experiments	123
A.8.1	Layer-wise Equivariance Baselines	123
A.8.2	Subnetwork Equivariance Analysis	124
A.8.3	Model List	124
A.8.4	Alternative End-to-End Equivariance Metrics	125
A.8.5	LEE for Additional Transformations	127
A.8.6	Rotated MNIST Finetuning	127
B	Appendix: Transfer from Text to Numerical Data	128
B.1	Detailed method and hyperparameters	128
B.1.1	Input scaling	128

B.1.2	Validation tuning	128
B.1.3	Likelihood adjustment for GPT Models	129
B.2	Benchmarking details and extended results	130
B.2.1	Darts datasets	130
B.2.2	Monash datasets	132
B.2.3	Informer datasets	134
B.2.4	Synthetic datasets	135
B.2.5	Darts full probabilistic results	135
B.2.6	Informer datasets with extended horizon	135
B.2.7	Monash dataset visualizations	136
B.2.8	Informer dataset visualizations	136
B.3	Simplicity bias experiments	136
B.3.1	Full synthetic predictions	139
B.4	GPT-4	140
B.5	Multimodal Text Understanding of Time Series	141
C	Appendix: Language Modeling for 3D Crystal Data	156
C.1	Training Details	156
C.1.1	Numerical Formatting	156
C.1.2	Training with Stochastic Prompts	157
C.1.3	Extended Materials Project Dataset	157
C.1.4	Training Hyperparameters and Details	158
C.1.5	Role of Text Pretraining	158
C.2	Model Evaluation	160
C.2.1	Evaluation with ML potentials and DFT	160
C.2.2	Stability Checks and Percentages	161

C.2.3	Trade-Offs in Sampling	161
C.2.4	“Hallucination” Examples	162
C.2.5	Increase in Perplexity under Transformation (IPT)	165
C.2.6	Diversity and Novelty Calculation	165
C.2.7	Sampling Speed	166
C.3	Template Method Baseline	167
D	Appendix: Challenges in Applying Language Models to Numerical Data	172
D.1	Conditional vs Unconditional Modeling	172
D.2	Learning Speedup from Loss Masking	173
D.3	GNN Training Details	175
D.4	xVal Ablations	175
D.5	Scaling Experiment	175
D.6	Hyperparameter Settings	176
D.6.1	From-scratch models	176
D.6.2	Fine-tuned models	176
D.7	MAE Numbers with Standard Errors	178
D.8	RASP-L Programs for All Pairwise Distances	179
	Bibliography	182

List of Figures

2.1	The common perception in physics-informed machine learning is that increased performance is the result of complex biases. We find, however, that simpler implicit biases (such as second-order structure) often account for almost all of the improvement over baselines.	7
2.2	Left: The degree of energy violation ($ \hat{H} - H / \hat{H} H $) on test rollouts as a function of rollout relative error ($\ \hat{z} - z\ /\ \hat{z}\ \ z\ $) across different environments and random seeds. Both HNNs and NeuralODEs are scattered around the line $x = y$. Conditioned on the rollout performance, whether or not the model is Hamiltonian has little impact on the energy violation. Right: Energy violation on test trajectories is plotted as a function of the time T of the rollout, with the shaded regions showing 1 standard deviation in log space taken across 5 random seeds and the test trajectories.	12
2.3	Left: Test rollout error as a function of the regularization weighting in the loss. Even at an optimally chosen symplectic regularization strength, the benefit to model generalization is negligible. Right: Test rollout error plotted against the final value of the symplecticity error for the regularized models. For systems with more than a couple degrees of freedom, symplecticity error is <i>negatively</i> correlated with the quality of predictions.	14

2.4	<p>Left: NODE model with and without second-order structure (encoding $dq/dt = v$). Right: HNN models with and without second-order structure. Models with the SO bias significantly outperform those that do not. Error bars show standard error across 5 seeds.</p>	15
2.5	<p>Left: Log rollout error for NODEs with second order bias and HNNs trained chain pendulums, where the analytic form of the Hamiltonian is simpler than the vector field. Right: MechanicsNNs and HNNs trained on spring pendulums, which have Hamiltonians and vector fields of similar complexity. HNNs outperform NODE with second order bias on systems that use non-Cartesian coordinates. Error bars show standard error across 5 seeds.</p>	16
2.6	<p>Comparing the performance on damped systems. The NODE + SO matches the performance of a SymODEN with a fraction of the parameters and compute. HNNs without forcing terms encode the wrong inductive biases and thus fit the data poorly. Error bars denote standard error across 5 seeds. . .</p>	18
2.7	<p>HNNs perform very poorly on complex dynamics like OpenAI Gym Mujoco control systems. Biasing the model towards Hamiltonian dynamics makes it difficult to fit the training data. Simply imposing second-order structure on a NODE is much more effective. Error bars show standard error across 4 seeds.</p>	20
3.1	<p>Non-linearities generate new high-frequency harmonics.</p>	25

3.2	(Left): The Lie derivative measures the equivariance of a function under continuous transformations, here rotation. (Center): Using the Lie derivative, we quantify how much each layer contributes to the equivariance error of a model. Our analysis highlights surprisingly large contributions from nonlinearities, which affects both CNNs and ViT architectures. (Right): Translation equivariance as measured by the Lie derivative correlates with generalization in classification models, across convolutional and non-convolutional architectures. Although CNNs are often noted for their intrinsic translation equivariance, ViT and Mixer models are often more translation equivariant than CNN models after training.	27
3.3	Lie derivatives can be computed using automatic differentiation. We show how a Lie derivative for continuous rotations can be implemented in PyTorch [173]. The implementation in our experiments differs slightly, for computational efficiency and to pass second-order gradients through <i>grid_sample</i> . . .	29

- 3.4 Contributions to equivariance shown cumulatively by layer, in the order the layers occur in the network. **Left:** Convolutional architectures. In all the CNNs, much of the equivariance error comes from downsampling and nonlinearities. **Middle-Left:** Non-convolutional architectures. The initial patch embedding, a strided convolution, is the largest contributor for the ViTs and Mixers. The rest of the error is distributed uniformly across other nonlinear operations. **Middle-Right:** ResNet-50 across different transformations as a percentage. Despite being designed for translation equivariance, the fraction of equivariance error produced by each layer is almost identical for other affine transformations, suggesting that aliasing is the primary source of equivariance error. **Right:** Comparing LEE with alternative metrics for translation equivariance. Using integer translations misses key contributors to equivariance errors, such as activations, while using fractional translations can lead to radically different outcomes depending on choice of normalization (N or \sqrt{N}). LEE captures aliasing effects and has minimal design decisions. 31
- 3.5 Equivariance metrics evaluated on the ImageNet test set. **Left:** Non-LEE equivariance metrics display similar trends to Figure 3.2, despite using larger, multi-pixel transformations. **Right:** Norm of rotation and shear Lie derivatives. Across all architectures, models with strong generalization become more equivariant to many common affine transformations. Marker size indicates model size. Error bars show one standard error over test set images used in the equivariance calculation. 33

3.6	Case studies in decreasing translational equivariance error, numbered left-to-right. 1: Blur-Pool [245], an architectural change to improve equivariance, decreases the equivariance error but by less than can be accomplished by improving the training recipe or increasing the scale of model or dataset. 2-3: Increasing the number of parameters for a fixed model family (here ViTs [163] and EfficientNets [198]). 4: Increasing the training dataset size for a ResMLP Big [209] model. 5: Changing the training recipe for ResNeXt-50 [231] with improved augmentations [225] or SSL pretraining [235]. Error bars show one standard error over images in the Lie derivative calculation.	35
3.7	Models are less equivariant on test data and becoming decreasingly equivariant as the data moves away from the training manifold. As examples of data with similar distributions, we show equivariance error on the ImageNet train and test sets as well as CIFAR-100. As examples of out-of-distribution data, we use two medical datasets (which often use Imagenet pretraining), one for Histology [127] and one for Retinopathy [123].	37
4.1	We propose LLMTime , a method for time series forecasting with large language models (LLMs) by encoding numbers as text and sampling possible extrapolations as text completions. LLMTime can outperform many popular time series methods without any training on the target dataset (i.e. zero-shot). The performance of LLMTime also scales with the power of the underlying base model. Notably, models that undergo alignment (e.g. RLHF) do not follow the scaling trend. For example, GPT-4 demonstrates inferior performance to GPT-3.	44

- 4.2 **Left:** Autoregressive models over sequences of digits act like hierarchical softmax distributions over the corresponding numbers. When combined with uniform distributions in each discrete bucket, distributions over strings can become expressive distributions over continuous domains. **Right:** Using simple autoregressive models (e.g. RNNs) trained on a string representation of numbers, we can fit complex distributions that can be challenging for other methods, such as heavy-tailed or multimodal distributions. A simple autoregressive model can match or outperform well-known methods for density estimation, such as Gaussian mixture models (GMMs) or binning with a fixed resolution, as measured by Wasserstein distance between samples. 47
- 4.3 Careful tokenization is important for good forecasting with LLMs. Using the Australian Wine dataset from Darts [107], with values [151, 167, ..., 267], we show the tokenization used by GPT-3 [30] and LLaMA-2 [206] and the corresponding effect on forecasting performance. Added spaces allow GPT-3 to create one token per digit, leading to good performance. LLaMA-2, on the other hand, tokenizes digits individually, and adding spaces hurts performance. 49

4.4	Example predictions on exchange rate (left), ETTm2 (a sequence of electricity transformer temperature readings, center), and weather (right) for NHiTS [33], Autoformer [230], and last value predictions, as well as the historical standard deviation of the change from the last observed value. On the exchange [130] and ETTm2 [254] datasets there is minimal structure to be exploited except on very short horizons, and forecasts tend to under-perform simple baselines. On semi-structured datasets like weather, models can capture some overall structure, such as NHiTS accurately predicting the final values in the forecasting window, but are still only on par with naive predictions. From these plots we see why probabilistic evaluation is necessary and point estimates are insufficient.	52
4.5	LLMTime with base model GPT-3 or LLaMA-2 70B has the best or second best aggregated performance on several deterministic time series benchmarks [81, 107, 254] while being entirely zero-shot. Collectively, these benchmarks comprise 29 individual datasets with diverse sources, lengths, and noise levels. For Monash MAE numbers, established results are reported on unnormalized data, so we normalize values before aggregating (Appendix B.2.2). The informer datasets are multivariate, and we predict each covariate independently with LLMTime (Appendix B.2.3). GPT-3 evaluation on the Informer datasets was skipped because of the cost of API queries. Error bars show standard errors over the individual datasets in each benchmark.	53

4.6	Extended experiments on the Darts datasets. Left: Example probabilistic forecasts with baseline negative log likelihood per dimension (NLL/D). LLMs easily extrapolate trends (e.g. AirPassengers) and reproduce local patterns when data is noisy (e.g. GasRateCO2). Center: When using probabilistic metrics like NLL and CRPS, LLMTime outperforms all baselines, including PromptCast [234], a competing LLM method. Error bars show standard errors over datasets with Darts. Right: LLMTime is much more sample efficient than competing methods. While the performance of other methods degrades rapidly when we restrict them to a fraction of the original training set, LLMTime can assign high likelihood with only a few examples.	55
4.7	Evaluation on a collection of short univariate time series recorded after GPT-3’s training cutoff date. We compare the performance of our GPT-3 predictor against popular time series models. Predicted median and 10-90th percentile intervals are shown for GPT-3 given the context, and we compare test negative log likelihoods. GPT-3 continues to be competitive with or outperforms the baselines on all of the tasks, from in-context learning alone. This result reinforces our belief that GPT-3’s performance is not due to memorization of the test data.	56

4.8	LLMs can find low complexity explanations of the data, enabling them to zero-shot extrapolate numerical sequences. Left: GPT-3 likelihoods favor solutions from symbolic regression (PySR [50]) that balance training loss and complexity, leading to good generalization. Right: GPT-3 predicted median and 10-90th percentile prediction interval are shown given 140 timesteps of context. On the right of each time series, we show the log likelihoods compared to the ARIMA and TCN time series models. Overall, GPT-3 performs considerably better than the baselines, though composition and exponential growth are more challenging for the models (Appendix B.3.1).	59
4.9	Left: Time series forecasting performance (NLL/D and CRPS on Darts [107]) improves with reasoning performance of the underlying model LLM, as judged by accuracy on the Massive Multitask Language Understanding (MMLU) benchmark [105]. Displayed results are for all GPT-3, LLaMA [208], and LLaMA-2 [206] base models. Center: GPT-4 performs worse than GPT-3. Right: Forecasting performance (NLL/D and CRPS on Darts) appears to be negatively affected by alignment procedures (e.g. instruction tuning and RLHF) in general. LLaMA-2 chat models typically perform worse than the corresponding base model. Error bars show standard errors over individual datasets.	61

4.10	<p>Left: <code>LLMTime</code> can handle missing values without interpolation by denoting missingness with text (e.g. ‘NaN’). For baseline methods we perform linear interpolation and then fit the model as usual. <code>LLMTime</code> assigns higher log likelihood to datasets preprocessed with added ‘NaN’s than baseline methods assign to interpolated datasets. Forecasting performance, as judged by CRPS, is competitive between <code>LLMTime</code> and alternative methods that use explicit interpolation. Filled area shows standard error over individual datasets and 3 random seeds. Right: LLMs can be used to answer questions about time series data posed as text. We show GPT-4’s accuracy at predicting the function that generated the time series, obtained using chain-of-thought prompting.</p>	63
5.1	<p>Overview of our approach to materials generation with large language models. Using string formatted crystals and task-specific prompting, we enable unconditional stable materials generation, text-condition materials generation, and structural infilling. Base LLaMA-2 models are fine-tuned on a database of known inorganic materials [147] using low-rank adapters.</p>	68
5.2	<p>(left) We convert the crystal lattice, atom identities, and atom positions into strings. The model is trained to generate a structures conditioned on the text prompt, which might contain additional information about the composition, properties, or a starting structure to modify. (right) Energy above hull (E_{hull}) quantifies the stability of a material. A crystal with $E_{\text{hull}} < 0.1$ will be energetically favorable both in its structure and composition.</p>	72
5.3	<p>A sample with “hallucinated” element identities (Ln).</p>	77

5.4	Stability of LLaMA samples compared to CDVAE [232]. Fine-tuned LLaMA-2 70B generates a higher rate of metastable ($\hat{E}_{\text{hull}} < 0.1$) and stable materials than CDVAE, using estimates of \hat{E}_{hull} from both M3GNet [37] and VASP [95]. Because of computational cost, we only run VASP on structures predicted to be stable by M3GNet. Stable materials generated by LLaMA are also more diverse (as quantified by Matminer featurization [219]) than stable samples from CDVAE. We include sampled stable structures, shown as (2,2,2) supercells, which display a high-degree of regularity and understanding of three-dimensional space.	78
5.5	We compare LLaMA-2 models with CDVAE in their ability to generate novel and diverse samples as well as their overall speed. (left) We calculate diversity and novelty using a featurization of structure and composition (as in Table 5.1). Diversity is calculated as pairwise distance in feature space, while novelty quantifies the percentage of inputs that are far from the training set (Appendix C.2.6). All metrics are calculated only for samples that were already judged to be metastable. LLaMA-2 models often generate more diverse samples than CDVAE, and achieve similar overall rates of novelty. Interestingly, structural novelty is lower in larger models, while compositional novelty is higher. (right) We compare the time required to generate 10,000 samples from each model. We run LLaMA-2 models with the largest feasible batch size on one A100 GPU (Appendix C.2.7). While the largest LLaMA model is computationally expensive, smaller language models are very fast, especially when we consider both sampling speed and rate of stability.	79

5.6	Text-conditional generation and infilling of existing structures with fine-tuned LLMs. (left) Including composition or property information (sampled from a hold-out set) in the text prompt leads to a high rate of samples with the desired composition/property (space group or stability). We bin stability as $\hat{E}_{\text{hull}} < 0.1$ (metastable) and $\hat{E}_{\text{hull}} > 0.1$ (unstable) for simplicity. Complex formulas and space groups challenge the model, but the samples are correct at a rate that facilitates practical use. We also show the rate of samples that both satisfy the condition and are predicted to be metastable by M3GNet. (right) Using the infilling prompt we can select mutations to existing materials. LLaMA-2 70B proposes a distribution over elements, which we constrain using knowledge of atom radii and charge interactions. We sample mutations with temperature τ and relax the results structure with M3GNet. When we apply this mutation procedure, we obtain more stable materials per mutation, with negligible changes to the overall diversity of the stable materials.	81
5.7	Translation invariance on test data and ability to generate stable materials increase in proportion. Larger models learn invariances from augmentations more effectively during training, likely as a result of their preference for abstract and compressible patterns.	82
6.1	To study language models applied to property prediction, we break each into its constituent parts. We deconstruct property prediction by studying building block operations from linear algebra and understand language model performance through the lens of architecture, tokenization, and pretraining. .	86

6.2	Encoder-decoder architectures have theoretical advantages over decoder-only architectures on our numerical tasks, but we find in practice the difference is minor. In our encoder-decoder models, layers are split equally between the encoder and decoder. A task name with ‘+’ indicates a holdout of unseen matrix shape—a harder test of generalization. We include quantized numerical operations as baselines. <code>16 bit</code> refers to a quantized operation with an 8-bit mantissa and 8-bit exponent. <code>20 bit</code> has a 10 bit mantissa and 10 bit exponent. We do not provide results for a quantized eigenvalue solver because PyTorch does not provide an easy mechanism for constructing one.	94
6.3	(Top) Degree of invariance (permutation or rotation error) strongly correlates with the ability to fit the task (MAE) across several model sizes, tokenization methods, and training runs. Results are displayed with both axes log-scaled. For rotation invariance, we only study tasks on 3D structures. Shading is a 95% confidence interval for the regression.	97
6.4	(Left) We train causal transformers with different tokenization schemes and witness a significant advantage from learning a continuous prediction head. By contrast, differences between discrete tokenization schemes (digits vs. chunks) are inconsistent with multi-digit schemes performing better on some tasks and worse on others. (Center) Using a continuous prediction head leads to higher invariance at smaller model sizes. For discrete methods, larger models are required to learn invariance. Numbers are the geometric mean over tasks, and shading denotes a 95% confidence interval. (Right) We present a scaling experiment on the matrix product task. We train 5 different model sizes, for both standard digit tokenization and a continuous prediction head. Shading denotes parameter count, while FLOPs are a function of parameter count and gradient steps. Additional details are provided in section D.5. . . .	98

6.5	We can model numerical data as strings or as vectors of continuous values. Each approach has unique costs and benefits.	100
6.6	We compare our small language models trained from scratch with finetuned versions of large text-pretrained models. Text-pretrained models perform worse on every task except matrix products, which might benefit from the pretrained model’s additional capacity and ability to model high-dimensional outputs.	103
A.1	Switching from l2 to l1 loss can improve rollout error slightly, but doesn’t impact the ordering of the models. The other elements of the experimental setup are identical to above. Error bars show one standard deviation. . . .	114
A.2	On the additional systems from Finzi, Wang, and Wilson [69], we can observe the effect of second order structure, compared with NODE and HNN baselines. As before, second order structure seems to account for much of the difference between NODE and HNN models. Error bars show one standard deviation.	115

- A.3 LEE calculated over the subnetworks of a ResNet50. Specifically a subnetwork is constructed between the input and every intermediate representation in the network’s computation graph. We use batch normalization of the outputs to make the output scale of different subnetwork comparable. For visual clarity, layer types are broken across the left and right plots, which share the same axes. Similar to the pattern observed in Figure 3.4, we see a rapid increase in equivariance error in the early layers of the network, followed by many smaller increases later in the network. Unlike in our layerwise decomposition, comparison across layer types is challenging in this setting because layers have significantly different outputs. For example, comparing activations with preactivations is complicated by the ReLUs acting as contractions of the input and having potentially many zeroed values. 125
- A.4 **(Left)**: Extending Figure 3.5 we show the Lie derivate norm for hyperbolic rotation, brightening, and stretch transformations. We observe that more accurate models are also more equivariant to hyperbolic rotations and to brighten transformation, to a more limited extent. In the case of hyperbolic rotations, this result is surprising, as nothing has directly encouraged this equivariance. One possible explanation is decreased aliasing in models with higher accuracy. Marker size indicates model size. Error bars show one standard error over the images use to evaluate the Lie derivative. **(Right)**: Cumulative mean and standard error of the estimator (computed for translations on a ResNet-50). 127

B.1	Median predictions of LLMTIME (GPT-3) and NLLs from LLMTIME (GPT-3 and LLaMA-2 70B) for every dataset within Darts [107]. The shaded area shows the 10th to 90th quantiles of the distribution over samples. LLMTIME consistently obtains better likelihood values than the baselines and often makes surprisingly accurate forecasts by effectively extrapolating trend and periodic components.	136
B.2	Aggregated and non-aggregated MAE numbers for LLMTIME (LLaMA-2 70B base model) and baselines on the Informer datasets. Overall LLMTIME performs well in aggregate for a zero-shot method, but its performance is highly variable, being the best method on some datasets and the worst on others. The relative performance of LLMTIME is slightly diminished for a longer prediction horizon, but LLMTIME is still very competitive with the best methods in aggregate. Error bars show two standard deviations in the error over datasets.	137
B.3	LLMTIME (GPT-3 base model) median predictions on at most 4 randomly chosen series per Monash dataset.	138
B.4	LLMTIME (LLaMA-2 70B base model) median predictions on 4 randomly chosen series per Informer dataset.	139
B.5	LLMTIME median predictions on all synthetic datasets using GPT-3 as a base model. The hyperparameters used are described in Appendix B.2.4.	140
B.6	GPT-4 extrapolations on synthetic data (10-90th percentiles shaded). GPT-4 is able to identify and extrapolate the pattern for each of the deterministic time series, but sometimes behaves erratically.	141
B.7	GPT-4 extrapolations on real (DARTS) time series (10-90th percentiles shaded). The extrapolations are plausible but worse than GPT-3, and the uncertainties tend to be more poorly calibrated making for a high CRPS.	142

C.1	Validity and rate of stability depend on sampling hyper-parameters. Lowering the temperature or restricting the nucleus size leads to significant improvements in validity/stability but incurs a cost to coverage of a held-out test set (recall). Fine-tuned LLaMA-2 70B displays the best trade-off between coverage and stability, generating materials that are both stable and diverse.	162
D.1	(Left) We include ablations on xVal to explore the effect of working with discrete versus continuous inputs and the corresponding loss functions. (Right) To understand the performance of xVal, we perform an ablation the output and input with discrete tokens to understand if continuous inputs or continuous outputs (continuous loss) is the origin of improved performance. Both ablations hurt performance, but continuous inputs appear to be more helpful than continuous outputs.	176

List of Tables

3.1	Our finetuned MAE is competitive with several architectures explicitly engineered to encode rotation invariance on RotMNIST, where rotation invariance is clearly crucial to generalization.	36
4.1	Multivariate results with varying prediction lengths. Bolded results indicate the best performing model, and italics the second best. In all cases simple statistics of the input data to the model are either the first or second best performing models in terms of both MSE and MAE accuracy. Historical Inertia (HI) [53] was also introduced as a trivial baseline but has worse performance than our constants.	51
5.1	Following prior work [232], we evaluate fine-tuned LLaMA-2 models using validity, which captures physical constraints, as well as coverage and property metrics, which capture alignment between the ground truth and sampling distribution. We add stability checks, which count the percentage of samples estimated to be stable by M3GNet [37] and DFT [95] (details in Appendix C.2.2). LLaMA models generate a high percentage of both valid and stable materials.	77
6.1	Comparison of popular and state-of-the-art approaches for predicting HOMO on QM9.	89

6.2	Digit order has a negligible effect on relative error. We report geometric mean across tasks with standard errors.	97
6.3	GNNs outperform LMs on the energy prediction task (from coordinates) and benefit from equivariance.	99
D.1	MAE (\downarrow) values for training with and without masking, both from scratch and finetuning.	172
D.2	Hyperparameter values for model architecture for from-scratch language model training runs.	176
D.3	Hyperparameter values for from-scratch language model training runs. . . .	177
D.4	Hyperparameter values for fine-tuning language model training runs.	177
D.5	MAE values for different tasks and tokenization methods. Standard errors are calculated from 200 data points from each task.	177

List of Appendices

A	Appendix: Deconstructing Learned Symmetries	109
B	Appendix: Transfer from Text to Numerical Data	128
C	Appendix: Challenges in Applying Language Models to Numerical Data	172

1 | INTRODUCTION

The role of inductive bias in deep learning emerged as a central point of tension during my PhD (2020-2025). Early successes like AlphaFold-2 [122] demonstrated how incorporating domain knowledge through architectural constraints—specifically symmetries and invariances from protein physics—could achieve breakthrough performance. This aligned with the geometric deep learning paradigm [29], which advocated for building domain structure directly into neural architectures. However, concurrent work in language models, particularly GPT-3 [30], revealed that massive scaling with minimal architectural changes could yield remarkable capabilities. This trend soon spread to computer vision, where generic architectures like Vision Transformers [63] and CLIP [176] began outperforming domain-specific approaches. By the end of my PhD, even protein structure prediction showed signs of this shift: AlphaFold-3 [1] achieved state-of-the-art results by replacing hand-designed geometric constraints with learned data augmentations. This evolution raises fundamental questions about the nature of scientific data: Do scientific domains truly require specialized architectural constraints, or will sufficient data and scale eventually obviate the need for manual inductive biases? This thesis explores the boundary between learnable and fundamental structural constraints across different scientific domains.

Inductive bias encompasses the assumptions built into a model before exposure to data. When searching a hypothesis space (\mathcal{H}) for a model (\mathcal{M}) that fits the data, these prior assumptions $P(\mathcal{M})$ guide selection between models with similar training performance, typically

improving generalization or sample efficiency. In deep learning, these biases often manifest through architecture design—convolutional neural networks encode translation equivariance, while transformers process pairwise relationships.

The emergence of larger datasets has shifted attention toward how different architectural choices scale with data volume. While domain-specific architectures often excel with limited data, simpler architectures with fewer constraints can match or exceed their performance given sufficient training examples. This dynamic has practical implications: when different approaches achieve similar results, considerations like computational efficiency and framework compatibility often favor simpler architectures. The rise of transfer learning has further complicated this tradeoff, as pretraining on large datasets can implicitly encode useful biases that previously required careful architecture design.

Large language models (LLMs) exemplify this shift toward data-driven architecture design. Using a relatively simple framework—transformers trained on next-token prediction—LLMs demonstrate consistent performance improvements with increased training data across diverse tasks. Their ability to process many data types through text encoding has reduced the need for domain-specific architectures, shifting research focus from architecture design toward dataset curation and evaluation. This makes LLMs an ideal case study for examining when and how large-scale pretraining can replace traditional architectural constraints.

This thesis examines the interplay between architectural constraints and data-driven learning through five investigations. The first two chapters reveal how energy conservation in Hamiltonian Neural Networks and translation equivariance in CNNs can be achieved through weaker architectural constraints combined with appropriate training data. The third and fourth chapters demonstrate how LLMs can tackle scientific tasks—including time series extrapolation and crystal structure prediction—often surpassing domain-specific approaches. The final chapter bridges these perspectives, analyzing why LLMs struggle with molecular

property prediction and providing insights into the roles of architecture, tokenization, and pretraining in scientific applications.

Part I

Distilling Architecture Constraints for Learning Symmetric Functions

In this section, I present two case studies examining how neural network architecture design influences symmetry learning. Both studies challenge conventional wisdom about the relationship between architectural constraints and learned symmetries.

The first study investigates energy conservation in neural networks trained on dynamical systems. While Hamiltonian neural networks (HNNs) are designed to enforce energy conservation through their architecture, our analysis reveals unexpected findings. Through systematic ablation studies, we demonstrate that HNNs do not conserve energy in the ways commonly assumed by practitioners. Furthermore, we show that comparable performance can be achieved using a simpler, more flexible inductive bias, suggesting that architectural constraints may be unnecessarily restrictive.

The second study examines how neural networks learn symmetries in image classification tasks, focusing on equivariance under affine transformations. We introduce a novel metric called the Local Equivariance Error (LEE), derived from differential geometry's Lie derivative. Using this metric, we demonstrate that Vision Transformers (ViTs) trained with data augmentation can achieve stronger translational equivariance than Convolutional Neural Networks (CNNs), despite CNNs having translational equivariance embedded in their architecture. This finding challenges the assumption that architectural constraints are necessary for learning important symmetries.

This research was conducted in collaboration with Marc Finzi, Sam Stanton, Micah Goldblum, and Andrew Gordon Wilson, resulting in two publications: "Deconstructing the Inductive Biases of Hamiltonian Neural Networks" (ICLR 2022) and "The Lie Derivative for Measuring Learned Equivariance" (ICLR 2023).

2 | DECONSTRUCTING THE INDUCTIVE BIASES OF HAMILTONIAN NEURAL NETWORKS

2.1 INTRODUCTION

The inductive biases of convolutional neural networks, such as translation equivariance, locality, and parameter sharing, play a key role in their generalization properties. Other biases have similarly guided the development of deep models for other domains like graphs and point clouds. Yet since models often encode multiple biases simultaneously, it can be challenging to identify how each contributes to generalization in isolation. By understanding which inductive biases are essential, and which are merely ancillary, we can simplify our models, and improve performance. For example, Wu et al. [229] demonstrate that Graph Convolutional Networks can be radically simplified to mere logistic regression by removing nonlinearities, leading to dramatic gains in computational efficiency, while retaining comparable accuracy.

Hamiltonian Neural Networks (HNNs) [85] have emerged as a leading approach for modeling dynamical systems. These dynamics models encode physical priors and outperform alternative Neural ODE approaches [41]. In the spirit of Wu et al. [229], we seek to identify

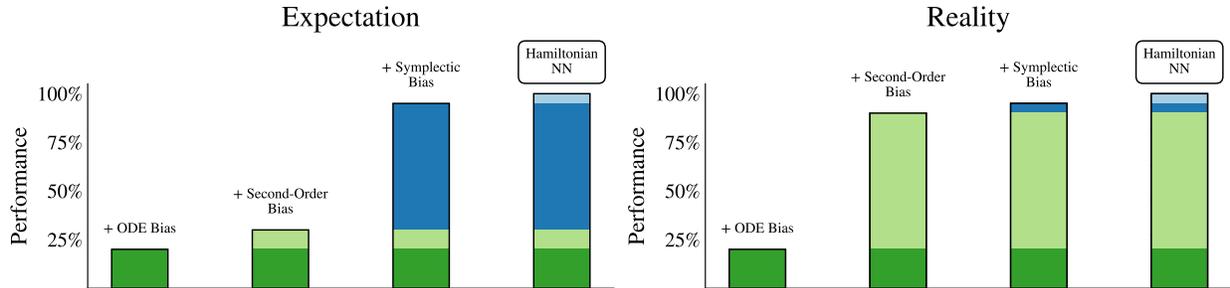


Figure 2.1: The common perception in physics-informed machine learning is that increased performance is the result of complex biases. We find, however, that simpler implicit biases (such as second-order structure) often account for almost all of the improvement over baselines.

the critical components of HNNs. Since Lagrangian models (LNNs) [51, 150] share the same structure and inductive biases as HNNs, we focus on HNNs where energy conservation and symplecticity are more explicit.

HNNs encode a number of inductive biases that help model physical systems:

1. **ODE bias:** HNNs model derivatives of the state rather than the states directly.
2. **Second-order (SO) bias:** HNNs model changes in position through changes in velocity.
3. **Energy conservation bias:** HNNs conserve their learned energy function.
4. **Symplectic bias:** HNNs dynamics are *symplectic*: phase space areas are conserved, and the vector field has a Hamiltonian structure.

In this chapter we theoretically and empirically examine the role of these biases. In contrast to conventional wisdom, we find that the generalization benefit of HNNs is not explained by their symplectic or energy-conserving properties, but rather by their implicit second-order bias. We highlight these findings in Figure 2.1. Abstracting and extracting out this second-order bias, we show how to improve the performance of Neural ODEs, empowering applications when HNNs assumptions are violated as is often the case. Code for our experiments can be found at: <https://github.com/ngruver/decon-hnn>.

2.2 RELATED WORK

PHYSICS-INSPIRED MODELS AND ENERGY CONSERVATION Since Greydanus, Dzumba, and Yosinski [85], many researchers have sought to extend the HNN approach to make it more general or applicable to systems that break energy conservation. Jin et al. [121], Li et al. [141], Tong et al. [204], and Xiong et al. [233] propose methods for creating neural networks that preserve symplectic structure directly. Zhong, Dey, and Chakraborty [251], and later Desai et al. [60], Li et al. [143], and Lee, Trask, and Stinis [138] propose models with additional capacity for changes to the system energy. In our analysis, we compare against approaches that add additional capacity for changes in energy, although our approach is fundamentally different — we attempt to remove unnecessary biases rather than add complexity.

PHYSICS-INSPIRED MODELS FOR CONTROL A large and rapidly expanding body of work explores how to use physics-based biases in dynamics models with controls or contacts [39, 44, 92, 93, 110, 151, 252, 253] intended for application in model-based planning. Especially relevant to our evaluations, Alvarez, Roşca, and Fălcuţescu [5] models MuJoCo [202] trajectories with a numerically integrated neural network but does not explore other physics-inspired inductive biases.

ANALYZING PHYSICS-BASED INDUCTIVE BIASES Karniadakis et al. [125] and Liu et al. [149] propose conceptual frameworks for deep learning with physics-based inductive biases, but present minimal empirical analysis of common design decisions. Zhong, Dey, and Chakraborty [250] compare many approaches to physics-inspired deep learning, with results that parallel some findings here. The contribution of our work, however, is not simply benchmarking but also an actionable theory of HNNs’ success. In spirit, our work is similar to Botev et al. [22], which also critically examines the role of physics-based priors in dynamics

models, though their focus is on learning latent space dynamics, while we focus on temporal models in observation space.

2.3 BACKGROUND

We consider dynamical systems described by ordinary differential equations (ODEs) which determine how the system evolves over time. Even with high order derivatives, these systems can be arranged into the form $\frac{dz}{dt} = F(z, t)$ for $z \in \mathbb{R}^n$. If the dynamics F are time-independent, then the ODE can be understood as a *vector field*, specifying at each point where the next state will be.

Neural ODEs (NODEs) [41] parametrize a vector field with a neural network and learn the dynamics directly from observed trajectories. A NODE dynamics model \hat{F}_θ is rolled out from the initial condition z_0 with ODE integration, $\hat{z}_t = \text{ODESolve}(z_0, \hat{F}_\theta, t)$, and fit to trajectory data by minimizing an L2 loss $L(\theta) = \sum_{t=1}^T \|\hat{z}_t - z_t\|^2$ between the predicted and observed trajectories, \hat{z}_t and z_t .

Like NODEs, HNNs also model dynamical systems as a parameterized vector field,

$$\frac{dz}{dt} = J\nabla\hat{H}_\theta \text{ with } J = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix}, \quad (2.1)$$

where \hat{H}_θ is a neural network with scalar output [85].

This differential equation expresses Hamiltonian dynamics where p and q are the canonical positions and momenta,

$$z = \begin{bmatrix} q \\ p \end{bmatrix} \text{ and } \frac{d}{dt} \begin{bmatrix} q \\ p \end{bmatrix} = \begin{bmatrix} \frac{\partial \hat{H}}{\partial p} \\ -\frac{\partial \hat{H}}{\partial q} \end{bmatrix}. \quad (2.2)$$

It is common practice to also explicitly define $H = T + V$, where T and V are the kinetic and potential energy [69, 92, 253]. The assumption that the Hamiltonian is *separable* holds

for mechanical systems and allows for further simplification,

$$\hat{H}_\theta(q, p) = \frac{1}{2} p^T M_\theta^{-1}(q) p + V_\theta(q), \quad (2.3)$$

where positive definite mass matrix M and scalar V are outputs of the neural network.

2.4 DECONSTRUCTING HAMILTONIAN NEURAL NETWORKS

In the following section we investigate to what extent commonly held beliefs about HNN properties actually explain the ability of HNNs to generalize. To separate out the different properties of these models, we select synthetic environments from Finzi, Wang, and Wilson [69] and Finzi, Welling, and Wilson [70] that are derived from a time independent Hamiltonian, where energy is preserved exactly. We use `kChainPendulum`, a k link pendulum in angular coordinates, and `kSpringPendulum`, a pendulum connected with k spring links in Cartesian coordinates.

We compute relative error between predicted states, \hat{z} , and ground truth, z , as $\|\hat{z} - z\|_2 / \|\hat{z}\|_2 \|z\|_2$ and between the predicted and ground truth Hamiltonian as $|\hat{H} - H| / |\hat{H}| |H|$. Following Finzi, Wang, and Wilson [69], we evaluate the performance of the model by computing the geometric mean of the relative error of the state over rollouts of length 20 times the size used at training, which more faithfully predicts downstream performance compared to other metrics like MSE [69]. The geometric mean of a function f over time T is given by $\exp(\frac{1}{T} \int_{t=0}^T \log f(t) dt)$.

2.4.1 ENERGY CONSERVATION

HNNs are commonly believed to be superior for energy conservation than comparable models [51, 85]. While there is some empirical evidence to support this claim, a precise

mathematical explanation for why this should be the case has not been established. Surprisingly, we find that conditioned on the trajectory reconstruction error, HNNs are no better at conserving the true energy of the system than unconstrained NeuralODE models.

Up to the numerical accuracy of an ODE solver, HNNs do conserve their own learned energy function \hat{H} (different from the true energy of the system H), due to basic properties of Hamiltonian mechanics,

$$\frac{d\hat{H}(\hat{z})}{dt} = \nabla\hat{H}^\top \frac{d\hat{z}}{dt} = \nabla\hat{H}^\top J \nabla\hat{H} = 0, \quad (2.4)$$

where the last equality follows from the antisymmetry of J . If the HNN has fit the data well, we might expect \hat{H} to be close to the true Hamiltonian H , and so if \hat{H} is conserved then it seems that H should be too. As we show in Appendix A.2.2, the problem with this argument is that we have no guarantees that \hat{H} and H are close even if we have fit the data well with low error in the rollouts or the dynamics. Since the dynamics error and the training rollout error depend only on gradients $\nabla\hat{H}$, while the gradients may be close, the differences between the two scalar functions can grow arbitrarily large.

In Figure 2.2 (left) we show that the degree of energy conservation is highly correlated with the rollout error of the model, regardless of the choice of architecture. While HNNs have better rollout performance than NeuralODEs, they are on the same regression line with Rollout Error \propto Energy Violation. The differences in energy violation are best predicted by the rollout error and *not* the architecture.

In Appendix A.2.1, we derive a bound that helps explain this behaviour. Given that the trajectories are in a bounded region of the phase space and that there is a fixed amount of error in the dynamics model, the energy violation grows at most linearly in time and the dynamics error. In Figure 2.2 (right) we demonstrate empirically that energy is *not*

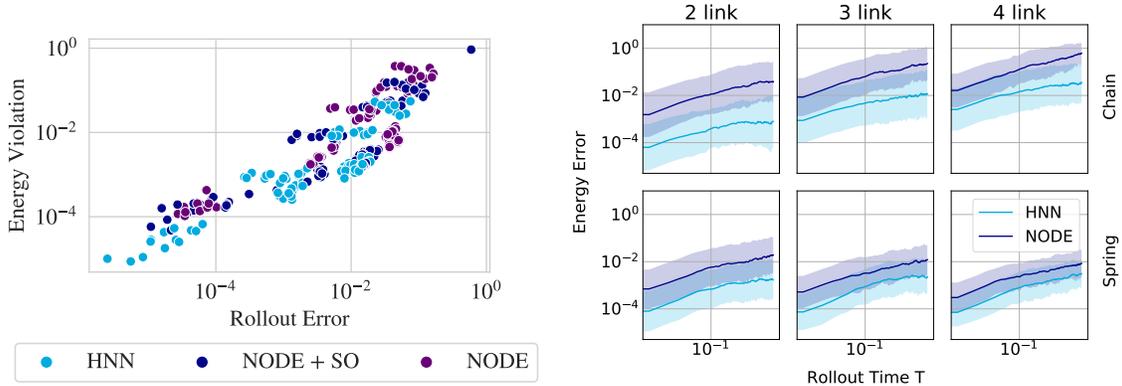


Figure 2.2: **Left:** The degree of energy violation ($|\hat{H} - H|/|\hat{H}| |H|$) on test rollouts as a function of rollout relative error ($\|\hat{z} - z\|/\|\hat{z}\| \|z\|$) across different environments and random seeds. Both HNNs and NeuralODEs are scattered around the line $x = y$. Conditioned on the rollout performance, whether or not the model is Hamiltonian has little impact on the energy violation. **Right:** Energy violation on test trajectories is plotted as a function of the time T of the rollout, with the shaded regions showing 1 standard deviation in log space taken across 5 random seeds and the test trajectories.

conserved as time progresses for HNNs.¹ In fact, the energy error of both NODE and HNN models grow linearly as our bound suggests. Although energy conservation may be helpful for generalization, the evidence does not indicate that HNNs are inherently better at conserving energy than NODEs, suggesting that the superior generalization of HNNs cannot be attributed to superior energy conservation.

2.4.2 SYMPLECTIC VECTOR FIELDS

A defining property of Hamiltonian mechanics is the fact that the dynamics are *symplectic*. If energy conservation does not explain the effectiveness of HNNs, the symplectic property of HNN dynamics may be the cause. Informally, one of the consequences of symplectic dynamics is that every part of the state space is equally attractive and repulsive. There are no sources where all nearby trajectories flow out from, or sinks where all nearby trajectories flow into, only saddle points and centers where the inflow and outflow is balanced. Symplectic integrators [139] make use of this property for more stable integration of

¹The energy violation is considerably larger than merely the numerical error associated with the solver.

Hamiltonian systems over very long timespans, and it is intuitive that enforcing this property in learned dynamics would have benefits also, at the very least for reducing the size of the hypothesis space.

More formally, symplecticity is the property that the J matrix (the symplectic form) is preserved by the dynamics (Equation 2.1). This condition can be expressed as a constraint on the Jacobian DF of the vector field $F(z) = \frac{dz}{dt}$ (here the derivative D maps from a function to its Jacobian). In terms of the Jacobian, symplecticity is the condition $DF^\top J + JDF = 0$ or equivalently $(JDF)^\top = JDF$ since $J^\top = -J$. The significance of this condition is that areas occupied by states in phase space (which have units of energy) are preserved by symplectic transformations. One consequence is that a volume of solutions in phase space will continue to occupy the same volume over time and will not be compressed or expanded. In other words, the vector field has 0 divergence: $\text{Tr}(DF) = 0$, which one can derive from the above expression.

On \mathbb{R}^n , it can be shown that all symplectic vector fields can be expressed as the dynamics of some Hamiltonian system, and vice versa:

$$F = J\nabla H \Leftrightarrow (JDF)^\top = JDF \tag{2.5}$$

To show the forward direction, one can simply substitute in Hamiltonian dynamics. It is clear that the symplecticity property is satisfied: using $J^2 = -I$, $JDF = JD(J\nabla H) = J^2\nabla^2 H = -\nabla^2 H$, and $-\nabla^2 H$ is a symmetric matrix. The reverse direction is less obvious; however, by Poincaré’s lemma if a vector field F satisfies $(JDF)^\top = JDF$ on \mathbb{R}^n , then there exists a Hamiltonian function H such that $F = J\nabla H$, which is shown in subsection A.2.3.

The equivalence of Hamiltonian dynamics and symplecticity allows us separate the unique properties of HNNs from other inductive biases that result indirectly from modeling F through H . Following Ghosh et al. [80], we can create a regularizer $\text{SymplecticError}(F) =$

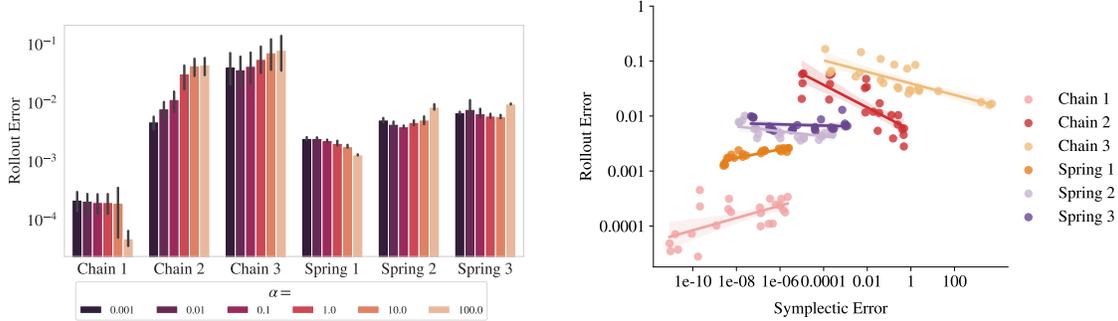


Figure 2.3: **Left:** Test rollout error as a function of the regularization weighting in the loss. Even at an optimally chosen symplectic regularization strength, the benefit to model generalization is negligible. **Right:** Test rollout error plotted against the final value of the symplecticity error for the regularized models. For systems with more than a couple degrees of freedom, symplecticity error is *negatively* correlated with the quality of predictions.

$\|(JDF)^T - JDF\|^2$ that directly measures the degree to which the symplectic property is violated. By parametrizing a NeuralODE and regularizing the symplectic error, we can enforce Hamiltonian structure while still directly modeling $\frac{dz}{dt} = F$ rather than H . Alongside an unregularized NeuralODE, we can isolate and evaluate the benefit of this Hamiltonian structure bias with a direct comparison.

Surprisingly, we find that the Hamiltonian structure bias, as enforced by the symplectic regularizer, provides no real benefit to the model’s ability to generalize over the long test rollouts (Figure 2.3 left). The achieved symplectic error Figure 2.3 (right) is not positively correlated with the final test rollout error of the model, and in some cases is even *negatively* correlated. Even when the symplectic error is very low and the symplecticity condition is enforced, there is no consistent improvement on the rollout generalization.

2.4.3 SECOND-ORDER STRUCTURE

If the superior performance of HNNs over NeuralODEs does not come from their better energy conservation properties, nor from the symplectic structure of the predicted vector field, what is the true cause?

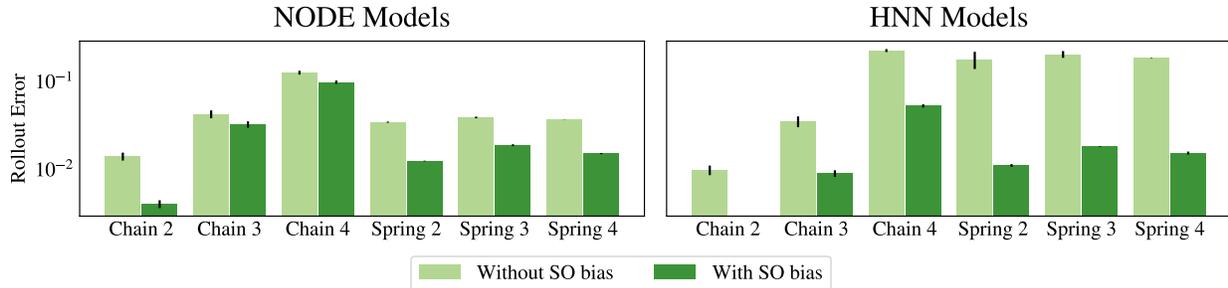


Figure 2.4: **Left:** NODE model with and without second-order structure (encoding $dq/dt = v$). **Right:** HNN models with and without second-order structure. Models with the SO bias significantly outperform those that do not. Error bars show standard error across 5 seeds.

In previous work, authors have used slightly different implementations of HNNs. One subtle improvement over the original work [85] comes from explicitly splitting the Hamiltonian as $\hat{H} = T + V = p^\top M(q)^{-1}p/2 + V(q)$ and modeling the mass matrix $M(q)$ and the potential $V(q)$ with separate neural networks rather than using a single neural network for \hat{H} [253]. This splitting enforces a strong assumption about the functional form of the Hamiltonian that applies to mechanical systems that makes it easier to learn and extrapolate. Through Hamilton's equations $\hat{F} = J\nabla\hat{H}$, this splitting is in fact specifying the relationship between position and momentum $\frac{dq}{dt} = \frac{\partial H}{\partial p} = M^{-1}(q)p$, and that forces can only affect $\frac{dp}{dt}$.

In fact, we can see that this assumption essentially leads to the second-order differential equation

$$\begin{bmatrix} \frac{dq}{dt} \\ \frac{dp}{dt} \end{bmatrix} = \begin{bmatrix} M^{-1}(q)p \\ -\frac{dV}{dq} \end{bmatrix} \implies \frac{d^2q}{dt^2} = \left(\frac{d}{dt} M^{-1}(q) \right) M(q) \frac{dq}{dt} - M^{-1}(q) \frac{dV}{dq} = A(q, \frac{dq}{dt}) \quad (2.6)$$

This second-order (SO) structure, $\begin{bmatrix} \frac{dq}{dt} \\ \frac{dv}{dt} \end{bmatrix} = \begin{bmatrix} v \\ A(q, v) \end{bmatrix}$, is a direct by-product of the separable Hamiltonian inductive bias, but is more general, applying to both conservative and non-conservative physical systems.

We can isolate the effect of this bias by directly observing its effect on both HNNs

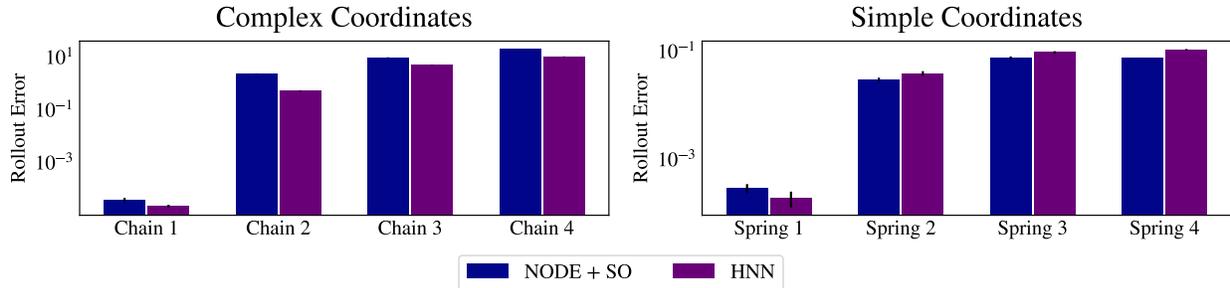


Figure 2.5: **Left:** Log rollout error for NODEs with second order bias and HNNs trained chain pendulums, where the analytic form of the Hamiltonian is simpler than the vector field. **Right:** MechanicsNNs and HNNs trained on spring pendulums, which have Hamiltonians and vector fields of similar complexity. HNNs outperform NODE with second order bias on systems that use non-Cartesian coordinates. Error bars show standard error across 5 seeds.

and NODEs. For HNNs the bias is made explicit in separable Hamiltonians, but not in the general case, when $H(q, p)$ is the direct output of the network instead of $V(q)$. We can design a NODE with second-order structure (NODE + SO) by setting $z = [q, p]$ with $p = Mv$ and $\frac{dq}{dt} = v, \frac{dp}{dt} = \tilde{A}_\theta(q, p)$, or equivalently just the second order equation $\frac{d^2q}{dt^2} = A_\theta(q, \frac{dq}{dt})$. Figure 2.4 shows the effect of second order structure on the test predictions of NODEs and HNNs. It is clear that this bias explains the superior performance of HNNs much more than other biases that are frequently given more credit. In fact, when we add this bias to NODE models, we see that their performance more closely matches HNNs than vanilla NODEs without creating a conservative or symplectic vector.

2.4.4 FUNCTIONAL COMPLEXITY

Adding second order structure to NODEs is always helpful, and matches the HNN performance for many of the systems. However, we see that there is still a gap for some systems, and curiously in each of these cases the system is described in a non-Cartesian coordinate system, such as with joint angles and Euler angles. Recall that with the symplecticity bias, we only found no benefit for enforcing that there *exists* a function \hat{H} such that $\hat{F} = J\nabla\hat{H}$ bias while parametrizing \hat{F} ; however, for HNNs this function not only exists, it is directly

parametrized by the neural net. If the function \hat{H} happens to be a simpler function to express and learn than ∇H , then representing the solution in this way can be beneficial.

For systems expressed in Cartesian coordinates like the spring pendulum, the mass matrix $M(q) = M$ is a constant, and so the gradients of $H = p^\top M p + V(q)$ are simple to express and learn. However, for systems with constraints such as the Chain pendulum, where states are typically expressed in angular coordinates, the mass matrix $M(q)$ will have a complicated form and that complexity will be magnified when taking the derivative. As an example, consider the Hamiltonian that an HNN must learn for the 2 link chain pendulum versus the vector field that a NODE + SO model must learn for this system (derived in Finzi, Wang, and Wilson [69], Appendix F.2). For the spring pendulum the functional complexity of the Hamiltonian and vector fields is comparable, while for the chain pendulum, the vector field contains many more terms.

Parameterizing such a system via its Hamiltonian simplifies the learning problem, and enables a neural network to converge more rapidly towards a plausible solution. This observation aligns with the insight in Finzi, Wang, and Wilson [69], which shows that changing the coordinate system to Cartesian dramatically simplifies the learning problem, at the expense of needing to enforce the constraints to the configuration space more directly.

Figure 2.5 shows the relative performance of the NODE+SO across the chain pendulum and spring pendulum environments. As we would now expect, the gap between the NODE+SO and HNN vanishes (and even favors NODE+SO) when complexity of the Hamiltonian and vector field are comparable.

2.5 DISTILLING THE MINIMAL INDUCTIVE BIASES

Perfectly energy-conserving systems are useful for analyzing the limiting behaviour of physics-informed networks, but in the vast majority of real world applications, we do not

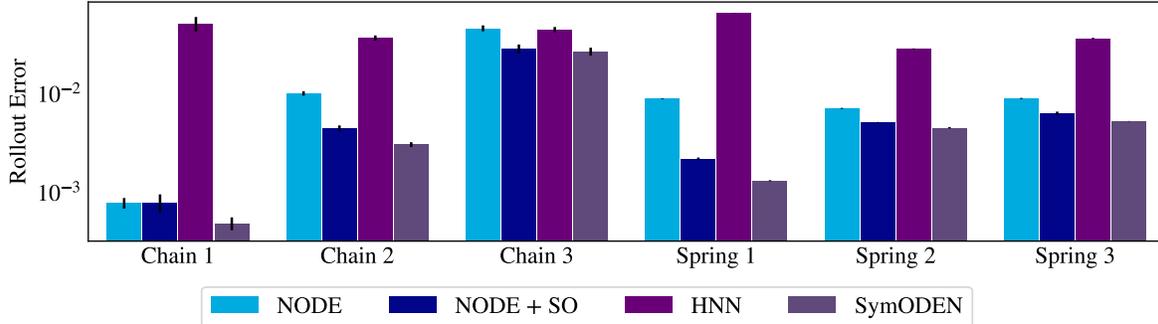


Figure 2.6: Comparing the performance on damped systems. The NODE + SO matches the performance of a SymODEN with a fraction of the parameters and compute. HNNs without forcing terms encode the wrong inductive biases and thus fit the data poorly. Error bars denote standard error across 5 seeds.

model closed systems. Energy is changed through contact with the environment (as in friction or drag) or an actor applying controls. In these cases, HNNs can be generalized by adding a forcing term

$$\begin{bmatrix} \frac{dq}{dt} \\ \frac{dp}{dt} \end{bmatrix} = \begin{bmatrix} \frac{\partial H}{\partial p} \\ -\frac{\partial H}{\partial q} \end{bmatrix} + \begin{bmatrix} 0 \\ g(q, p) \end{bmatrix} u \quad (2.7)$$

as in SymODEN [251, 253] and Desai et al. [60], where u is the control input, which can be fixed as constant in systems with drag or friction.

Though SymODEN can accommodate controls and damping, we show that simply using the bias of second-order dynamics is sufficient to achieve nearly the same performance with much less complexity. We demonstrate the matching performance on our n-body pendulum systems, augmented with drag, $\frac{dp}{dt} = -\frac{\partial H}{\partial q} - \lambda v$ (Figure 2.6).

HNNs are typically evaluated on relatively simple systems, like those considered in the previous sections. In principle, we would expect these results to extend to more complex systems that are governed by similar physical laws. In practice, however, there is little evidence to suggest that applying HNN methods to complex systems is easy or effective.

The MuJoCo physics simulator [202] is one such complex system that we would expect to benefit from HNN inductive biases. Gym Mujoco systems are heavily used in model-based

reinforcement learning (MBRL) literature, but the dynamics models commonly used are surprisingly simple (often just MLPs trained to predict the next change in state) [48, 217].

Given how much benefit other applications of deep learning have derived from specialized architectures, such as CNNs for computer vision [136], RNNs for NLP [158], or WaveNets for audio [168], we would expect analogous improvements to be possible in MBRL. Algorithms for MBRL are infamously sensitive to choice of prediction horizon, and one possible explanation is poor generalization caused by weak inductive biases [6, 119, 172]. Improving model design for mechanical systems has the potential to improve both the sample efficiency of MBRL algorithms and their robustness.

We train NODEs and HNNs on trajectories from several OpenAI Gym Mujoco environments [28]. Crucially, we compare NODEs endowed with second-order structure (NODE + SO) against pre-existing NODE and HNN models. Note that with fixed step size integrators, NODEs are equivalent to discrete transition models that predict the next state or delta directly with an MLP, and therefore our NODE baseline is representative of models commonly used in MBRL. See [section A.3](#) for implementation details.

In [Figure 2.7](#) we show that NODE + SO significantly outperforms baseline methods. Surprisingly HNNs underperform NODEs on all the systems we consider. Although the HNN could in principle learn the dynamics, in practice the bias towards Hamiltonian dynamics makes fitting the training data very difficult and provides no tangible benefit to generalization. This outcome is notably different from what we observe in toy tasks, where HNNs can fit non-conservative systems (e.g., pendulums with drag) with little difficulty.

In spirit, our results parallel the findings of Wu et al. [229] in the different setting of graph CNNs. We are able to distill the inductive biases of HNNs into a NODE + SO without losing performance on systems that HNNs perform well on, and, even more importantly, these reduced systems are much more capable of scaling to complex systems and larger datasets.

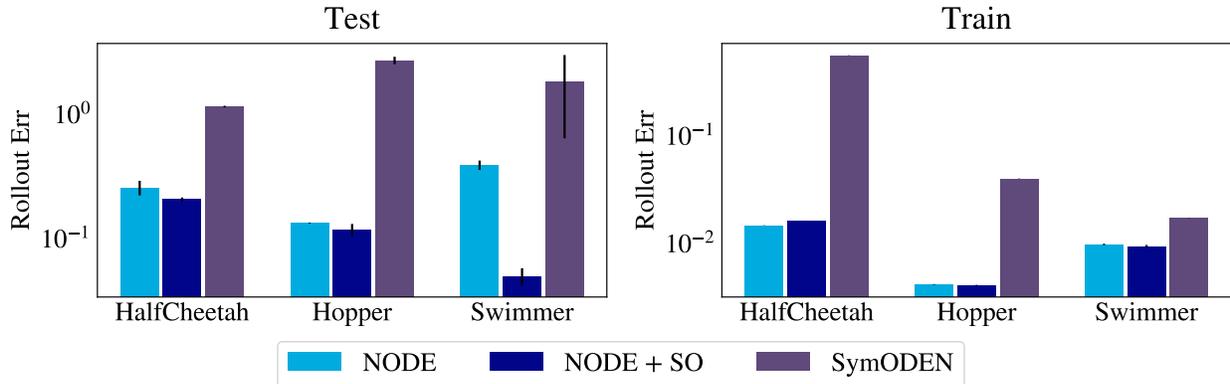


Figure 2.7: HNNs perform very poorly on complex dynamics like OpenAI Gym Mujoco control systems. Biasing the model towards Hamiltonian dynamics makes it difficult to fit the training data. Simply imposing second-order structure on a NODE is much more effective. Error bars show standard error across 4 seeds.

2.6 CONCLUSION

In this chapter, we deconstructed the inductive biases of highly performing HNN models into their component parts, a NeuralODE, symplecticity, conservation of the learned energy function, and second order structure. Contrary to conventional wisdom, the success of HNNs is not from their energy conservation or symplecticity, but rather from the assumption that the system can be expressed as a single second order differential equation. Stripping away the other components of an HNN, we are left with a model that is simpler, more computationally efficient, and less restrictive in that it can be directly applied to non-Hamiltonian systems. As a consequence, we are able to apply the resulting model to constructing transition models for the challenging Mujoco locomotion environments, with promising performance.

3 | THE LIE DERIVATIVE FOR MEASURING LEARNED EQUIVARIANCE

3.1 INTRODUCTION

Symmetries allow machine learning models to generalize properties of one data point to the properties of an entire class of data points. A model that captures translational symmetry, for example, will have the same output for an image and a version of the same image shifted a half pixel to the left or right. If a classification model produces dramatically different predictions as a result of translation by half a pixel or rotation by a few degrees it is likely misaligned with physical reality. Equivariance provides a formal notion of consistency under transformation. A function is equivariant if symmetries in the input space are preserved in the output space.

Baking equivariance into models through architecture design has led to breakthrough performance across many data modalities, including images [49, 214], proteins [122] and atom force fields [18, 74]. In computer vision, translation equivariance has historically been regarded as a particularly compelling property of convolutional neural networks (CNNs) [135]. Imposing equivariance restricts the size of the hypothesis space, reducing the complexity of

the learning problem and improving generalization [84].

In most neural networks classifiers, however, true equivariance has been challenging to achieve, and many works have shown that model outputs can change dramatically for small changes in the input space [15, 67, 181, 213]. Several authors have significantly improved the equivariance properties of CNNs with architectural changes inspired by careful signal processing [126, 245], but non-architectural mechanisms for encouraging equivariance, such as data augmentations, continue to be necessary for good generalization performance [225].

The increased prominence of non-convolutional architectures, such as vision transformers (ViTs) and mixer models, simultaneously demonstrates that explicitly encoding architectural biases for equivariance is not necessary for good generalization in image classification, as ViT models perform on-par with or better than their convolutional counterparts with sufficient data and well-chosen augmentations [63, 203]. Given the success of large flexible architectures and data augmentations, it is unclear what clear practical advantages are provided by explicit architectural constraints over learning equivariances from the data and augmentations. Resolving these questions systemically requires a unified equivariance metric and large-scale evaluation.

3.2 BACKGROUND

Equivariance provides a formal notion of consistency under transformation. A function $f : V_1 \rightarrow V_2$ is equivariant to transformations from a symmetry group G if applying the symmetry to the input of f is the same as applying it to the output

$$\forall g \in G : f(\rho_1(g)x) = \rho_2(g)f(x), \tag{3.1}$$

where $\rho(g)$ is the *representation* of the group element, which is a linear map $V \rightarrow V$.

The most common example of equivariance in deep learning is the translation equivariance of convolutional layers: if we translate the input image by an integer number of pixels in x and y , the output is also translated by the same amount, ignoring the regions close to the boundary of the image. Here $x \in V_1 = V_2$ is an image and the representation $\rho_1 = \rho_2$ expresses translations of the image. The translation *invariance* of certain neural networks is also an expression of the equivariance property, but where the output vector space V_2 has the trivial $\rho_2(g) = I$ representation, such that model outputs are unaffected by translations of the inputs. Equivariance is therefore a much richer framework, in which we can reason about representations at the input and the output of a function.

3.3 IMAGE CLASSIFICATION AND TRANSLATIONAL EQUIVARIANCE

3.3.1 CONTINUOUS SIGNALS AND ALIASING

CONTINUOUS SIGNALS The inputs to classification models are discrete images sampled from a *continuous* reality. Although discrete representations are necessary for computers, the goal of classification models should be learning functions that generalize in the real world. It is therefore useful to consider an image as a function $h : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ rather than a discrete set of pixel values and broaden the symmetry groups we might consider, such as translations of an image by vector $\mathbf{b} \in \mathbb{R}^2$, rather than an integer number of pixels.

Fourier analysis is a powerful tool for understanding the relationship between continuous signals and discrete samples by way of frequency decompositions. Any $M \times M$ image, $h(\mathbf{x})$, can be constructed from its frequency components, H_{nm} , using a $2d$ Fourier series, $h(\mathbf{x}) = \frac{1}{2\pi} \sum_{n,m} H_{nm} e^{2\pi i \mathbf{x} \cdot [n,m]}$ where $\mathbf{x} \in [0, 1]^2$ and $n, m \in [-M/2, -M/2 + 1, \dots, M/2]$, the bandlimit defined by the image size.

ALIASING Aliasing occurs when sampling at a limited frequency f_s , for example the size of an image in pixels, causes high frequency content (above the Nyquist frequency $f_s/2$) to be converted into spurious low frequency content. Content with frequency n is observed as a lower frequency contribution at frequency

$$\text{Alias}(n) = \left\{ \begin{array}{ll} n \bmod f_s & \text{if } (n \bmod f_s) < f_s/2 \\ (n \bmod f_s) - f_s & \text{if } (n \bmod f_s) > f_s/2 \end{array} \right\}. \quad (3.2)$$

If a discretely sampled signal such as an image is assumed to have no frequency content higher than f_s , then the continuous signal can be uniquely reconstructed using the Fourier series and have a consistent continuous representation. But if the signal contains higher frequency content which gets aliased down by the sampling, then there is an ambiguity and exact reconstruction is not possible.

ALIASING AND EQUIVARIANCE Aliasing is critically important to our study because it breaks equivariance to continuous transformations like translation and rotation. When a continuous image is translated its Fourier components pick up a phase:

$$h(\mathbf{x}) \mapsto h(\mathbf{x} - \mathbf{b}) \implies H_{nm} \mapsto H_{nm} e^{-2\pi i \mathbf{b} \cdot [n, m]}.$$

However, when an aliased signal is translated, the aliasing operation A introduces a scaling factor:

$$H_{nm} \mapsto H_{nm} e^{-2\pi i (b_0 \text{Alias}(n) + b_1 \text{Alias}(m))}$$

In other words, aliasing causes a translation *by the wrong amount*: the frequency component H_{nm} will effectively be translated by $[(\text{Alias}(n)/n)b_0, (\text{Alias}(m)/m)b_1]$ which may point in a different direction than \mathbf{b} , and potentially even the opposite direction. Applying shifts to an aliased image will yield the correct shifts for true frequencies less than the Nyquist but

incorrect shifts for frequencies higher than the Nyquist. Other continuous transformations, like rotation, create similar asymmetries.

Many common operations in CNNs can lead to aliasing in subtle ways, breaking equivariance in turn. Zhang [245], for example, demonstrates that downsampling layers causes CNNs to have inconsistent outputs for translated images. The underlying cause of the invariance is aliasing, which occurs when downsampling alters the high frequency content of the network activations. The $M \times M$ activations at a given layer of a convolutional network have spatial Nyquist frequencies $f_s = M/2$. Downsampling halves the size of the activations and corresponding Nyquist frequencies. The result is aliasing of all nonzero content with $n \in [M/4, M/2]$. To prevent this aliasing, Zhang [245] uses a local low pass filter (Blur-Pool) to directly remove the problematic frequency regions from the spectrum.

While studying generative image models, Karras et al. [126] unearth a similar phenomenon in the point-wise nonlinearities of CNNs. Imagine an image at a single frequency $h(\mathbf{x}) = \sin(2\pi\mathbf{x} \cdot [n, m])$. Applying a nonlinear transformation to h creates new high frequencies in the Fourier series, as illustrated in [Figure 3.1](#).

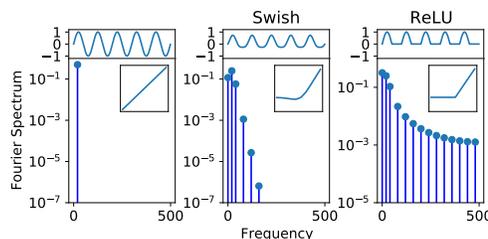


Figure 3.1: Non-linearities generate new high-frequency harmonics.

These high frequencies may fall outside of the bandlimit, leading to aliasing. To counteract this effect, Karras et al. [126] opt for smoother non-linearities and perform up-sampling before calculating the activations.

3.4 RELATED WORK

While many papers propose architectural changes to improve the equivariance of CNNs [126, 222, 245], others focus purely on measuring and understanding how equivariance can emerge through learning from the training data [140]. Olah et al. [165], for example, studies

learned equivariance in CNNs using model inversions techniques. While they uncover several fascinating properties, such as rotation equivariance that emerges naturally without architectural priors, their method is limited by requiring manual inspection of neuron activations. Most relevant to our work, Bouchacourt, Ibrahim, and Morcos [23] measure equivariance in CNNs and ViTs by sampling group transformations. Parallel to our findings, they conclude that data and data augmentation play a larger role than architecture in the ultimate equivariance of a model. Because their study is limited to just a single ResNet and ViT architecture, however, they do not uncover the broader relationship between equivariance and generalization that we show here.

Many papers introduce consistency metrics based on sampling group transformations [23, 126, 245], but most come with significant drawbacks. When translations are sampled with an integer number of pixels [23, 245], aliasing effects will be completely overlooked. As a remedy, [126] propose a subpixel translation equivariance metric (EQ- T_{frac}) that appropriately captures aliasing effects. While this metric is a major improvement, it requires many design decisions not required by LEE, which has relatively few hyperparameters and seamlessly breaks down equivariance across layers. Relative to these other approaches, LEE offers a unifying perspective, with significant theoretical and practical benefits.

3.5 THE LIE DERIVATIVE

3.5.1 MATHEMATICAL FORMULATION

The Lie derivative gives a general way of evaluating the degree to which a function f violates a symmetry. To define the Lie derivative, we first consider how a symmetry group element can transform a function by rearranging [Equation 3.1](#):

$$\rho_{21}(g)[f](x) = \rho_2(g)^{-1}f(\rho_1(g)x).$$

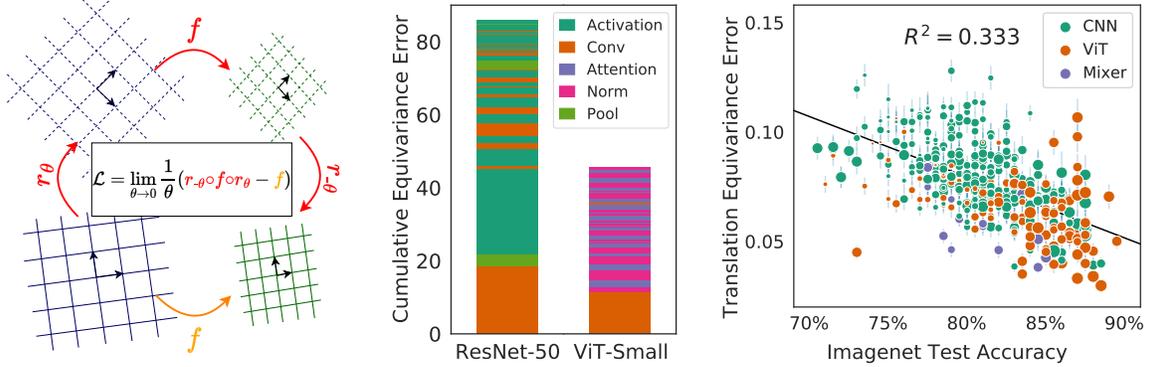


Figure 3.2: **(Left):** The Lie derivative measures the equivariance of a function under continuous transformations, here rotation. **(Center):** Using the Lie derivative, we quantify how much each layer contributes to the equivariance error of a model. Our analysis highlights surprisingly large contributions from non-linearities, which affects both CNNs and ViT architectures. **(Right):** Translation equivariance as measured by the Lie derivative correlates with generalization in classification models, across convolutional and non-convolutional architectures. Although CNNs are often noted for their intrinsic translation equivariance, ViT and Mixer models are often more translation equivariant than CNN models after training.

The resulting linear operator, $\rho_{21}(g)[\cdot]$, acts on the vector space of functions, and $\rho_{21}(g)[f] = f$ if the function is equivariant. Every continuous symmetry group (Lie group), G , has a corresponding vector space (Lie algebra) $\mathfrak{g} = \text{Span}(\{X_i\}_{i=1}^d)$, with basis elements X_i that can be interpreted as vector fields $\mathbb{R}^n \rightarrow \mathbb{R}^n$. For images, these vector fields encode infinitesimal transformations $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ over the domain of continuous image signals $f : \mathbb{R}^2 \rightarrow \mathbb{R}^k$. One can represent group elements $g \in G$ (which lie in the connected component of the identity) as the *flow* along a particular vector field Φ_Y^t , where $Y = \sum_i a_i X_i$ is a linear combination of basis elements. The flow $\Phi_Y^t(x_0)$ of a point x_0 along a vector field Y by value t is defined as the solution to the ODE $\frac{dx}{dt} = Y(x)$ at time t with initial value x_0 . The flow Φ_Y^t smoothly parameterizes the group elements by t so that the operator $\rho_{21}(\Phi_Y^t)[\cdot]$ connects changes in the space of group elements to changes in symmetry behavior of a function.

The Lie derivative of the function f is the derivative of the operator $\rho_{21}(g)$ at $g =$

Identity = Φ_0 along a particular vector field Y ,

$$\mathcal{L}_Y(f) = \lim_{t \rightarrow 0} \frac{1}{t} (\rho_{21}(\Phi_Y^t)[f] - f) = \left. \frac{d}{dt} \right|_{t=0} \rho_{21}(\Phi_Y^t)[f]. \quad (3.3)$$

Intuitively, the Lie derivative measures the sensitivity of a function to infinitesimal symmetry transformations. This local definition of equivariance error is related to the typical global notion of equivariance error. As we derive in Appendix A.6.1, if $\forall i = 1, \dots, d : \mathcal{L}_{X_i}(f) = 0$ (and the exponential map is surjective) then $\forall g \in G : f(\rho_1(g)x) = \rho_2(g)f(x)$ and for all x in the domain, and vice versa. In practice, the Lie derivative is only a proxy for strict global equivariance. We note global equivariance includes radical transformations like translation by 75% of an image, which is not necessarily desirable. In section 3.7 we show that our local formulation of the Lie derivative can capture the effects of many practically relevant transformations.

The Lie derivative of a function with multiple outputs will also have multiple outputs, so if we want to summarize the equivariance error with a single number, we can compute the norm of the Lie derivative scaled by the size of the output. Taking the average of the Lie derivative over the data distribution, we define Local Equivariance Error (LEE),

$$\text{LEE}(f) = \mathbb{E}_{x \sim \mathcal{D}} \|\mathcal{L}_X f(x)\|^2 / \dim(V_2). \quad (3.4)$$

Mathematically, LEE also has an appealing connection to consistency regularization [13], which we discuss in Appendix A.7.1.

3.5.2 EFFICIENT IMPLEMENTATION

We provide a Python implementation of the Lie derivative calculation for rotations in Figure 3.3 as an example.

```

1 import torch.nn.functional as F
2 from torch.autograd.functional import jvp
3
4 def rotate(imgs, theta):
5     """ Rotate images by angle theta and interpolate """
6     m = [[torch.cos(theta), torch.sin(theta), 0],
7          [-torch.sin(theta), torch.cos(theta), 0]]
8     m = torch.tensor(m)[None].expand(imgs.shape[0], -1, -1)
9     return F.grid_sample(imgs, F.affine_grid(m, imgs.size()), True)
10
11 def rotation_lie_deriv(model, imgs):
12     """ Lie deriv. of model w.r.t. rotation, can be scalar/image """
13     def rotated_model(theta):
14         z = model(rotate(imgs, theta))
15         img_like = (len(z.shape) == 4) # more complex for ViT/Mixer
16         return rotate(z, -theta) if img_like else z
17     return jvp(rotated_model, torch.zeros(1, requires_grad=True))[-1]
18
19 def e_lee(model, imgs):
20     """ Expected equiv. error (E[|Lf|^2]/d_out) w.r.t. rotation """
21     return rotation_lie_deriv(model, imgs).pow(2).mean()

```

Figure 3.3: Lie derivatives can be computed using automatic differentiation. We show how a Lie derivative for continuous rotations can be implemented in PyTorch [173]. The implementation in our experiments differs slightly, for computational efficiency and to pass second-order gradients through `grid_sample`.

3.5.3 ALTERNATIVE METRICS

3.6 LAYER-WISE EFFECTS ON EQUIVARIANCE

Unlike alternative equivariance metrics, the Lie derivative decomposes naturally over the layers of a neural network, since it satisfies the chain rule. As we show in Appendix A.6.2, the Lie derivative of the composition of two functions $h : V_1 \rightarrow V_2$ and $f : V_2 \rightarrow V_3$ satisfies

$$\mathcal{L}_X(f \circ h)(x) = (\mathcal{L}_X f)(h(x)) + df|_{h(x)}(\mathcal{L}_X h)(x), \quad (3.5)$$

where $df|_{h(x)}$ is the Jacobian of f at $h(x)$ which we will abbreviate as df . Note that this decomposition captures the fact that intermediate layers of the network may transform in

their own way:

$$f(h(x)) \mapsto \rho_{31}(g)[f \circ g](x) = \rho_3(g)^{-1}f(\rho_2(g)\rho_2(g)^{-1}h(\rho_1(g)x)) = \rho_{32}(g)[f] \circ \rho_{21}(g)[h](x)$$

and the Lie derivatives split up accordingly.

Applying this property to an entire model as a composition of layers $\text{NN}(x) = f_{N:1}(x) := f_N(f_{N-1}(\dots(f_1(x))))$, we can identify the contribution that each layer f_i makes to the equivariance error of the whole. Unrolling [Equation 3.5](#), we have

$$\mathcal{L}_X(\text{NN}) = \sum_{i=1}^N df_{N:i+1} \mathcal{L}_X f_i. \tag{3.6}$$

Intuitively, the equivariance error of a sequence of layers is determined by the sum of the equivariance error for each layer multiplied by the degree to which that error is attenuated or magnified by the other layers (as measured by the Jacobian). We evaluate the norm of each of the contributions $df_{N:i+1} \mathcal{L}_X f_i$ to the (vector) equivariance error $\mathcal{L}_X(\text{NN})$ which we compute using autograd and stochastic trace estimation, as we describe in [Appendix A.6.3](#). Importantly, the sum of norms may differ from the norm of the sum, but this analysis allows us to identify patterns across layers and pinpoint operations that contribute most to equivariance error.

Subtle architectural details often prevent models from being perfectly equivariant. Aliasing can result from careless downsampling or from an activation function with a wide spectrum. In this section, we explore how the Lie derivative uncovers these types of effects automatically, across several popular architectures. We evaluate the equivariance of pre-trained models on 100 images from the ImageNet [\[59\]](#) test set.

Using the layerwise analysis, we can dissect the sources of translation equivariance error in convolutional and non-convolutional networks as shown in [Figure 3.4](#) (left) and (middle-left). For the Vision Transformer and Mixer models, we see that the initial conversion from

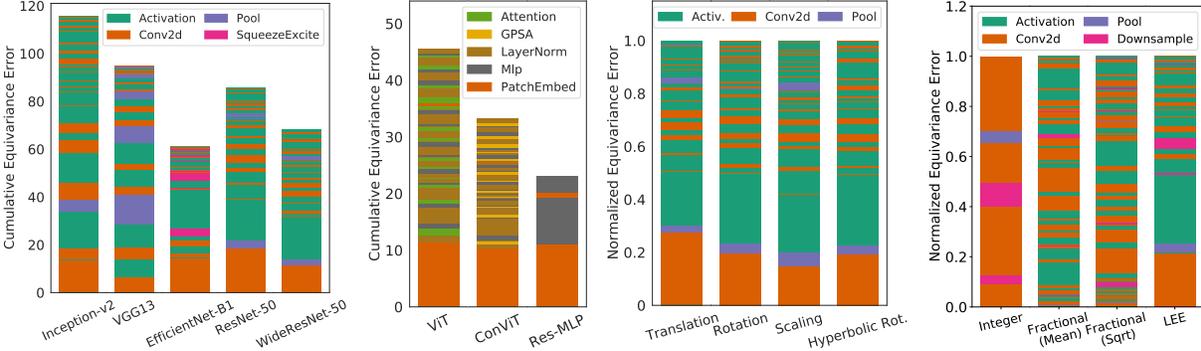


Figure 3.4: Contributions to equivariance shown cumulatively by layer, in the order the layers occur in the network. **Left:** Convolutional architectures. In all the CNNs, much of the equivariance error comes from downsampling and non-linearities. **Middle-Left:** Non-convolutional architectures. The initial patch embedding, a strided convolution, is the largest contributor for the ViTs and Mixers. The rest of the error is distributed uniformly across other nonlinear operations. **Middle-Right:** ResNet-50 across different transformations as a percentage. Despite being designed for translation equivariance, the fraction of equivariance error produced by each layer is almost identical for other affine transformations, suggesting that aliasing is the primary source of equivariance error. **Right:** Comparing LEE with alternative metrics for translation equivariance. Using integer translations misses key contributors to equivariance errors, such as activations, while using fractional translations can lead to radically different outcomes depending on choice of normalization (N or \sqrt{N}). LEE captures aliasing effects and has minimal design decisions.

image to patches produces a significant portion of the error, and the remaining error is split uniformly between the other nonlinear layers: LayerNorm, tokenwise MLP, and self-attention. The contribution from these nonlinear layers is seldom recognized and potentially counterintuitive, until we fully grasp the deep connection between equivariance and aliasing. In [Figure 3.4](#) (middle-right), we show that this breakdown is strikingly similar for other image transformations like rotation, scaling, and hyperbolic rotations, providing evidence that the cause of equivariance error is not specific to translations but is instead a general culprit across a whole host of continuous transformations that can lead to aliasing.

We can make the relationship between aliasing and equivariance error precise by considering the aliasing operation A defined in [Equation 3.2](#).

Theorem 3.1. *For translations along the vector $v = [v_x, v_y]$, the aliasing operation A intro-*

duces a translation equivariance error of

$$\|\mathcal{L}_v(A)(h)\|^2 = (2\pi)^2 \sum_{n,m} H_{nm}^2 \left(v_x^2(\text{Alias}(n) - n)^2 + v_y^2(\text{Alias}(m) - m)^2 \right),$$

where $h(\mathbf{x}) = \frac{1}{2\pi} \sum_{n,m} H_{nm} e^{2\pi i \mathbf{x} \cdot [n,m]}$ is the Fourier series for the input image h .

We provide the proof in Appendix A.7.2. The connection between aliasing and LEE is important because aliasing is often challenging to identify despite being ubiquitous [126, 245]. Aliasing in non-linear layers impacts all vision models and is thus a key factor in any fair comparison of equivariance.

As alternative equivariance metrics exist, it is natural to wonder whether they can also be used for layerwise analysis. In Figure 3.4 (right), we show how two equivariance metrics from Karras et al. [126] compare with LEE, highlighting notable drawbacks. (1) Integer translation equivariance completely ignores aliasing effects, which are captured by both LEE and fractional translations. (2) Though fractional translation metrics (EQ- T_{frac}) correctly capture aliasing, comparing the equivariance of layers with different resolutions ($C \times H \times W$) requires an arbitrary choice of normalization. This choice can lead to radically different outcomes in the perceived contribution of each layer and is not required when using LEE, which decomposes across layers. We provide a detailed description of the baselines in Appendix A.8.1.

3.7 TRENDS IN LEARNED EQUIVARIANCE

3.7.1 METHODOLOGY

We evaluate the Lie derivative of many popular classification models under transformations including $2d$ translation, rotation, and shearing. We define continuous transformations on images using bilinear interpolation with reflection padding. In total, we evaluate 410

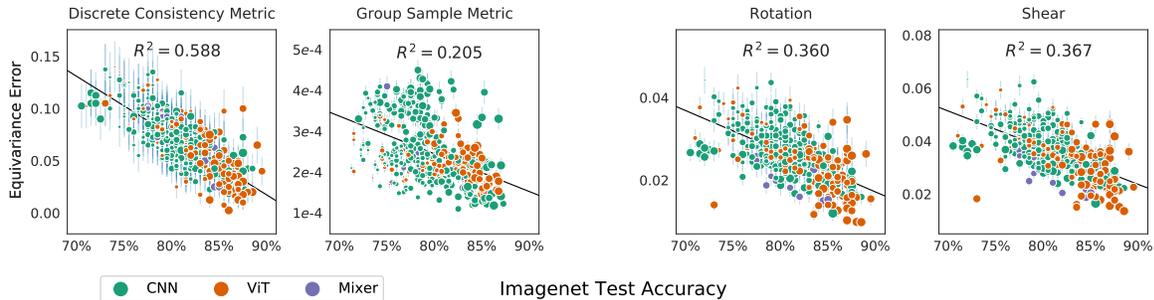


Figure 3.5: Equivariance metrics evaluated on the ImageNet test set. **Left:** Non-LEE equivariance metrics display similar trends to Figure 3.2, despite using larger, multi-pixel transformations. **Right:** Norm of rotation and shear Lie derivatives. Across all architectures, models with strong generalization become more equivariant to many common affine transformations. Marker size indicates model size. Error bars show one standard error over test set images used in the equivariance calculation.

classification models, a collection comprising popular CNNs, vision transformers, and MLP-based architectures [224]. Beyond diversity in architectures, there is also substantial diversity in model size, training recipe, and the amount of data used for training or pretraining. This collection of models therefore covers many of the relevant axes of variance one is likely to consider in designing a system for classification. We include an exhaustive list of models in the Appendix A.8.3.

3.7.2 EQUIVARIANCE ACROSS ARCHITECTURES

As shown in Figure 3.2 (right), the translation equivariance error (Lie derivative norm) is strongly correlated with the ultimate test accuracy that the model achieves. Surprisingly, despite convolutional architectures being motivated and designed for their translation equivariance, we find no significant difference in the equivariance achieved by convolutional architectures and the equivariance of their more flexible ViT and Mixer counterparts when conditioning on test accuracy. This trend also extends to rotation and shearing transformations, which are common in data augmentation pipelines [52] (in Figure 3.5 (right)). Additional transformation results included in Appendix A.8.5.

For comparison, we also evaluate the same set of models using two alternative equiv-

ariance metrics: prediction consistency under discrete translation [245] and expected equivariance under group samples [71, 116], which is similar in spirit to EQ- T_{frac} [126] (exact calculations in Appendix A.8.4). Crucially, these metrics are slightly less *local* than LEE, as they evaluate equivariance under transformations of up to 10 pixels at a time. The fact that we obtain similar trends highlights LEE’s relevance beyond subpixel transformations.

3.7.3 EFFECTS OF TRAINING AND SCALE

There are many architectural design choices that have been used to improve the equivariance of vision models, for example Zhang [245]’s Blur-Pool low-pass filter. We now investigate how equivariance error can be reduced with non-architectural design decisions, such as increasing model size, dataset size, or training method. Surprisingly, we show that equivariance error can often be significantly reduced without any changes in architecture.

In Figure 3.6, we show slices of the data from Figure 3.2 along a shared axis for equivariance error. As a point of comparison, in Figure 3.6 (left), we show the impact of the Blur-Pool operation discussed above on a ResNet-50 [245]. In the accompanying four plots, we show the effects of increasing model scale (for both ViTs and CNNs), increasing dataset size, and finally different training procedures. Although Zhang [245]’s architectural adjustment does have a noticeable effect, factors such as dataset size, model scale, and use of modern training methods, have a much greater impact on learned equivariance.

As a prime example, in Figure 3.6 (right), we show a comparison of three training strategies for ResNeXt-50 – an architecture almost identical to ResNet-50. We use Wightman, Touvron, and Jégou [225]’s pretrained model to illustrate the role of an improved training recipe and Mahajan et al. [153]’s semi-supervised model as an example of scaling training data. Notably, for a fixed architecture and model size, these changes lead to decreases in equivariance error on par with architectural interventions (BlurPool). This result is surprising when we consider that Wightman, Touvron, and Jégou [225]’s improved training recipe

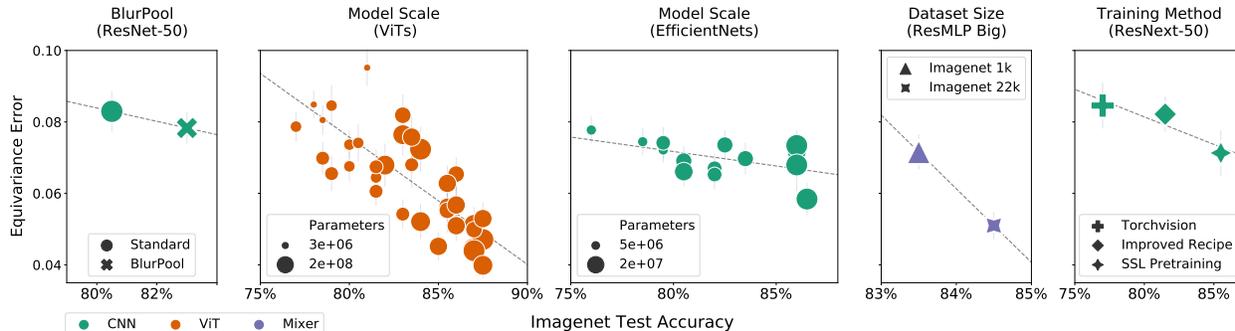


Figure 3.6: Case studies in decreasing translational equivariance error, numbered left-to-right. **1:** BlurPool [245], an architectural change to improve equivariance, decreases the equivariance error but by less than can be accomplished by improving the training recipe or increasing the scale of model or dataset. **2-3:** Increasing the number of parameters for a fixed model family (here ViTs [163] and EfficientNets [198]). **4:** Increasing the training dataset size for a ResMLP Big [209] model. **5:** Changing the training recipe for ResNeXt-50 [231] with improved augmentations [225] or SSL pretraining [235]. Error bars show one standard error over images in the Lie derivative calculation.

benefits significantly from Mixup [244] and CutMix [240], which have no obvious connection to equivariance. Similarly, Mahajan et al. [153]’s semi-supervised method has no explicit incentive for equivariance.

3.7.4 EQUIVARIANCE OUT OF DISTRIBUTION

From our analysis above, large models appear to learn equivariances that rival architecture engineering in the classification setting. When learning equivariances through data augmentation, however, there is no guarantee that the equivariance will generalize to data that is far from the training distribution. Indeed, Engstrom et al. [68] shows that carefully chosen translations or rotations can be as devastating to model performance as adversarial examples. We find that vision models do indeed have an *equivariance gap*: models are less equivariant on test data than train, and this gap grows for OOD inputs as shown in Figure 3.7. Notably, however, architectural biases do not have a strong effect on the equivariance gap, as both CNN and ViT models have comparable gaps for OOD inputs.

3.7.5 WHY AREN'T CNNs MORE EQUIVARIANT THAN ViTs?

Given the deep historical connection between CNNs and equivariance, the results in [Figure 3.5](#) and [Figure 3.7](#) might appear counterintuitive. ViTs, CNNs, and Mixer have quite different inductive biases and therefore often learn very different representations of data [178]. Despite their differences, however, all of these architectures are fundamentally constructed from similar building blocks—such as convolutions, normalization layers, and non-linear activations which can all contribute to aliasing and equivariance error. Given this shared foundation, vision models with high capacity and effective training recipes are more capable of fitting equivariances already present in augmented training data.

3.7.6 LEARNING ROTATION EQUIVARIANCE

We finally consider the extent to which large-scale pretraining can match strong architectural priors in a case where equivariance is obviously desirable. We fine-tune a state-of-the-art vision transformer model pretrained with masked autoencoding [102] for 100 epochs on rotated MNIST [222] (details in [Appendix A.8.6](#)). This dataset, which contains MNIST digits rotated uniformly between -180 and 180 degrees, is a common benchmark for papers that design equivariant architectures. In [Table 3.1](#) we show the test errors for many popular architectures with strict equivariance constraints alongside the error for our finetuned model. Surprisingly, the finetuned model achieves competitive test

Model	Test Error (%)
G-CNN [49]	2.28
H-NET [228]	1.69
ORN [257]	1.54
TI-Pooling [133]	1.2
Finetuned MAE	1.14
RotEqNet [154]	1.09
E(2)-CNN [222]	0.68

Table 3.1: Our finetuned MAE is competitive with several architectures explicitly engineered to encode rotation invariance on RotMNIST, where rotation invariance is clearly crucial to generalization.

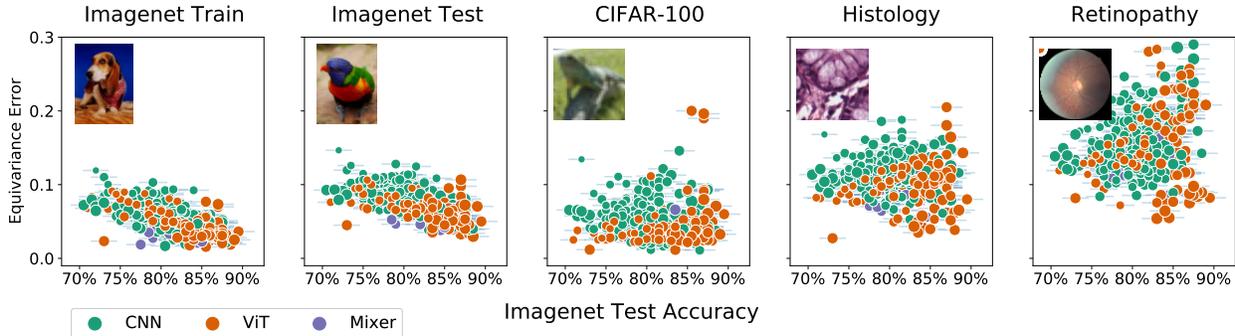


Figure 3.7: Models are less equivariant on test data and becoming decreasingly equivariant as the data moves away from the training manifold. As examples of data with similar distributions, we show equivariance error on the ImageNet train and test sets as well as CIFAR-100. As examples of out-of-distribution data, we use two medical datasets (which often use Imagenet pretraining), one for Histology [127] and one for Retinopathy [123].

accuracy, in this case a strong proxy for rotation invariance. Despite having relatively weak architectural biases, transformers are capable of learning and generalizing on well on symmetric data.

3.8 CONCLUSION

We introduced a new metric for measuring equivariance which enables a nuanced investigation of how architecture design and training procedures affect representations discovered by neural networks. Using this metric we are able to pinpoint equivariance violation to individual layers, finding that pointwise nonlinearities contribute substantially even in networks that have been designed for equivariance. We argue that aliasing is the primary mechanism for how equivariance to continuous transformations are broken, which we support theoretically and empirically. We use our measure to study equivariance learned from data and augmentations, showing model scale, data scale, or training recipe can have a greater effect on the ability to learn equivariances than architecture.

Many of these results are contrary to the conventional wisdom. For example, transformers can be more equivariant than convolutional neural networks after training, and can learn

equivariances needed to match the performance of specially designed architectures on benchmarks like rotated MNIST, despite a lack of explicit architectural constraints. These results suggest we can be more judicious in deciding when explicit interventions for equivariance are required, especially in many real world problems where we only desire approximate and local equivariances. On the other hand, explicit constraints will continue to have immense value when exact equivariances and extrapolation are required — such as rotation invariance for molecules. Moreover, despite the ability to learn equivariances on training data, we find that there is an equivariance *gap* on test and OOD data which persists regardless of the model class. Thus other ways of combating aliasing outside of architectural interventions may be the path forward for improving the equivariance and invariance properties of models.

DISCUSSION: DECONSTRUCTION BIASES

In Chapters 2 and 3, we demonstrated how models can learn symmetries from data and how models with less architectural constraints can perform better than models with less architectural constraints, even when the constraints encourage a known symmetry in the problem. Extrapolating from these results, should we then abandon architecture-based symmetry constraints entirely?

Probably not. In some cases it is beneficial to trade the complexity and limitations of symmetric architectures for the simplicity and generality of architectures with weaker inductive biases. In problems with abundant data, I predict that this will become more common. But in data-scarce settings, or in settings where marginal performance gains are game changing, the superior scaling laws of constrained architectures [26] justify their increased complexity. Notably, in some cases, the final wall clock time of training constrained architectures can be larger than that of unconstrained alternatives because of worse hardware utilization, as models like transformers have seen multiple waves of optimization for GPUs.

The intended take-away of this section is that often surprising design decisions that have the largest impact on the final symmetry properties of a model. For example, the energy conservation of HNNs is often misunderstood, and the invariance of CNNs is highly related to aliasing in the model, not just the theoretical properties of convolutions. In many cases, pinpointing the critical biases allows us to learn the property we care about more easily while also simplifying systems. Our mission is therefore not to eliminate inductive bias in some sense but to identify the critical and minimal set of inductive biases. In the next part of the thesis we take this mission a step further and probe how far we can go with just language models and large-scale text pretraining.

Part II

Sequence Modeling for Numerical Data: Transfer Success and Limitations

This section explores how language models, despite being trained exclusively on text data, can successfully transfer to numerical and scientific tasks. We focus on three distinct applications: time series forecasting, crystal structure prediction, and molecular property prediction. The first two cases reveal surprising capabilities that emerge from language models' pretraining on discrete sequences, despite these tasks being absent from their training data. In both time series forecasting and crystal structure prediction, we attribute the models' success to their inherent preference for simple functional relationships and their sophisticated probability estimation capabilities.

The applications demonstrate unexpected advantages of language models in scientific computing. Their sequence-based architecture enables seamless integration of textual information with numerical data, allowing researchers to frame traditionally distinct computational tasks within a unified modeling approach. However, these capabilities have limitations, as shown in our analysis of molecular property prediction, where we systematically examine how architectural choices, tokenization strategies, and pretraining methods affect model performance.

This research was conducted in collaboration with Marc Finzi, Shikai Qiu, Anuroop Sriram, Andrea Madotto, Dylan Sam, Larry Zitnick, Zachary Ulissi, and Andrew Gordon Wilson, resulting in publications at NeurIPS 2023 ("Large Language Models Are Zero-Shot Time Series Forecasters") and ICLR 2024 ("Fine-Tuned Large Language Models Generate Stable Inorganic Materials As Text"), with additional work currently under review.

4 | LARGE LANGUAGE MODELS ARE ZERO-SHOT TIME SERIES FORECASTERS

4.1 INTRODUCTION

Despite similarities with other sequence modeling problems, such as text, audio, or video, time series has two particularly challenging properties. Unlike video or audio, which typically have consistent input scales and sampling rates, aggregated time series datasets often comprise sequences from radically different sources, sometimes with missing values. Moreover, common applications of time series forecasting, such as weather or financial data, require extrapolating from observations that contain a tiny fraction of the possible information, making accurate point predictions nearly impossible and uncertainty estimation especially important. While large-scale pretraining has become a key element of training large neural networks in vision and text, enabling performance to scale directly with data availability, pretraining is not typically used for time series modeling, where there is no consensus unsupervised objective and large, cohesive pretraining datasets are not readily available. Consequently, simple time series methods (e.g. ARIMA [24], and linear models [241]) often outperform deep learning methods on popular benchmarks [108].

In this chapter, we demonstrate how large language models (LLM) can naturally bridge the gap between the simple biases of traditional methods and the complex representational learning and generative abilities of modern deep learning. In particular, we introduce an exceedingly simple method, **LLMTime**¹, to apply pretrained LLMs for continuous time series prediction problems, illustrated at a high level in [Figure 4.1](#). At its core, this method represents the time series as a string of numerical digits, and views time series forecasting as next-token prediction in text, unlocking the use of powerful pretrained models and probabilistic capacities, such as likelihood evaluation and sampling. To enable strong performance, we propose techniques to (1) effectively encode time series as a string of numerical digits and (2) adapt the discrete distributions of LLMs to continuous densities capable of modeling sophisticated multimodal distributions. Using these techniques, we find **LLMTime** can exceed or match purpose-built time series methods over a range of different problems in a *zero-shot* fashion, meaning that **LLMTime** can be used without any fine-tuning on the downstream data used by other models.

The zero-shot nature of **LLMTime** carries several natural advantages: (1) it facilitates the straightforward application of LLMs, eliminating the necessity for specialized knowledge of fine-tuning procedures and the substantial computational resources required for these procedures, as well as side-stepping access issues surrounding proprietary source code or APIs for LLM training or fine-tuning; (2) it is naturally suited to scenarios with limited data availability, where there is little information for training or fine-tuning; (3) by leveraging the broad pattern extrapolation capabilities of extensively pre-trained LLMs, it circumvents the extensive time, effort, and domain-specific expertise typically required for crafting dedicated time series models.

To understand the origins of **LLMTime**'s impressive performance, we investigate how LLMs express preferences for simple or repetitive sequences [82] and show that these biases are in

¹<https://github.com/ngruver/llmtime>

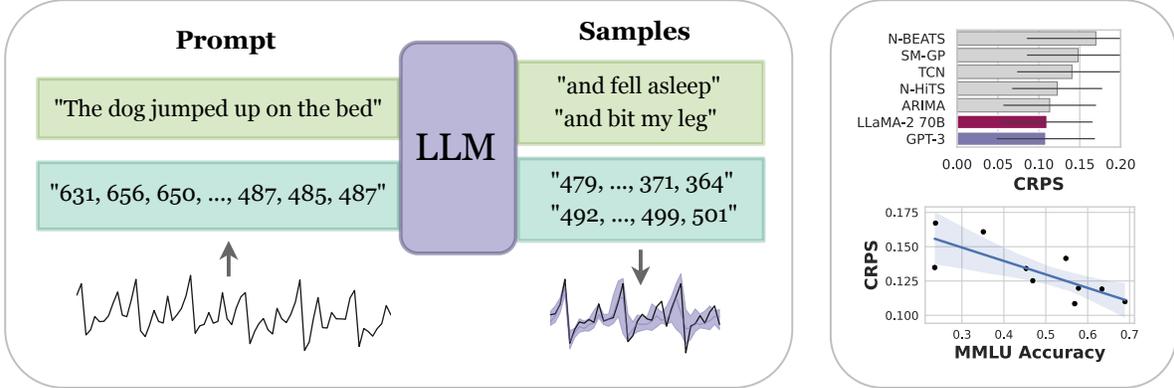


Figure 4.1: We propose LLMTIME, a method for time series forecasting with large language models (LLMs) by encoding numbers as text and sampling possible extrapolations as text completions. LLMTIME can outperform many popular time series methods without any training on the target dataset (i.e. zero-shot). The performance of LLMTIME also scales with the power of the underlying base model. Notably, models that undergo alignment (e.g. RLHF) do not follow the scaling trend. For example, GPT-4 demonstrates inferior performance to GPT-3.

fact compatible with the salient structure of time series, such as seasonality. Aside from these biases, LLMs also can naturally accommodate missing data, and express multimodal distributions, which is particularly useful for time series. We also show how LLMs enable appealing functionality, such as the ability to provide additional side information through prompting, and query the LLM to explain its predictions.

Finally, in addition to broadly compelling forecasting performance, we find performance tends to improve with scale, and the quality of point predictions also improves with the quality of the uncertainty representation. However, we also find GPT-4 has worse uncertainty calibration than GPT-3, likely due to interventions such as reinforcement learning by human feedback (RLHF).

4.2 BACKGROUND

LANGUAGE MODELING Language models are trained on a collection of sequences, $\mathcal{U} = \{U_1, U_2, \dots, U_i, \dots, U_N\}$, where $U_i = (u_1, u_2, \dots, u_j, \dots, u_{n_i})$ and each token, u_i , belongs to

a vocabulary \mathcal{V} . Large language models typically encode an autoregressive distribution, in which the probability of each token is only dependent on the previous tokens in the sequence, $p_\theta(U_i) = \prod_{j=1}^{n_i} p_\theta(u_j | u_{0:j-1})$. The parameters, θ , are learned by maximizing the probability of the entire dataset $p_\theta(\mathcal{U}) = \prod_{i=1}^N p_\theta(U_i)$. Every language model has an associated *tokenizer*, which breaks an input string into a sequence of tokens, each belonging to \mathcal{V} . Proper tokenization is extremely important, and small details can have surprisingly significant effects. The most common tokenization method for autoregressive language models is *byte-pair encoding* (BPE), which treats inputs like bit strings and assigns tokens based on the rate of occurrence in the training corpus, optimizing for shorter sequences of tokens on average. Sampling from a language model typically starts with a *prompt*, $u_{0:k}$, and proceeds sequentially using $p_\theta(u_j | u_{0:j-1})$, which is often preprocessed, for example through temperature scaling or nucleus sampling [111].

LARGE LANGUAGE MODELS Brown et al. [30] showed that increasing a language model’s parameter count and training data size leads to new capabilities such as *zero-shot generalization*, in which a model can perform a text-formatted task without training the model parameters on any task-specific data. Large language models, for example GPT-3 [30] or LLaMA-2 [206], accomplish this form of generalization through *in-context learning*, which identifies patterns in the language model’s prompt and extrapolates them through next-token prediction. Many authors have speculated that in-context learning emerges from a language model’s extensive compression of the input data [58, 82, 196]. Compression favors learning algorithms that operate over the input data with programmatic abstractions, for example, context-free grammars [4] or induction heads [166], which can implement copy-and-paste type operations for generating samples with highly structured syntax. In this work, we show that the zero-shot generalization abilities of LLMs and their preference for compressible patterns extend well beyond language understanding and can be used for time

series forecasting.

Zero-shot generalization has made LLMs significantly more useful as assistants, leading to the create of methods to align LLMs with human preferences and instructions, for example reinforcement learning from human feedback (RLHF) [171] and instruction tuning [221]. While key to modern LLMs products, alignment methods can also significantly affect the abilities and calibration of the underlying model [31, 169]. Here we show these methods can also affect forecasting ability.

4.3 RELATED WORK

4.4 CONTINUOUS DENSITIES WITH AUTOREGRESSIVE MODELS

The fact that LLMs can express flexible distributions over numbers is key for time series data. Uncertainty quantification is essential to forecasting, and typical approaches to representing uncertainty in time series can be limited by misspecification. For example, one common method for creating a probabilistic forecast is to fit a Gaussian or Laplace observation model. When the underlying data distribution is multimodal, both of these models will perform poorly. Methods like Gaussian mixture models (GMMs) solve the issue of multimodality but introduce additional challenges to optimization and model selection. We show that a language model is an underrated solution by training a small autoregressive model on a variety of one-dimensional distributions shown in [Figure 4.2](#) (right). These distributions come from an exponential random variable, a mixture of a uniform and a student-t distribution, and the heavy-tailed distribution of time series prediction residuals from an ARIMA model on the `MonthlyMilk` dataset [107]. We evaluate these fits quantitatively by computing Wasserstein distances, and compare to a Laplace observation model, a GMM

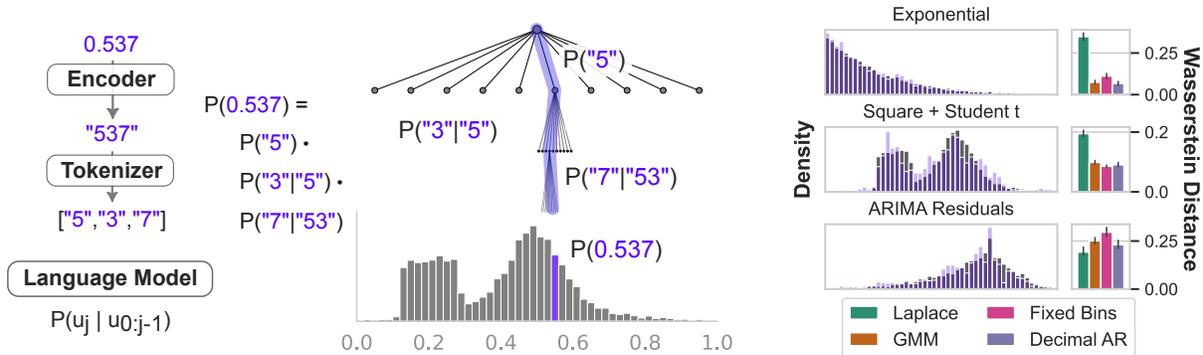


Figure 4.2: **Left:** Autoregressive models over sequences of digits act like hierarchical softmax distributions over the corresponding numbers. When combined with uniform distributions in each discrete bucket, distributions over strings can become expressive distributions over continuous domains. **Right:** Using simple autoregressive models (e.g. RNNs) trained on a string representation of numbers, we can fit complex distributions that can be challenging for other methods, such as heavy-tailed or multimodal distributions. A simple autoregressive model can match or outperform well-known methods for density estimation, such as Gaussian mixture models (GMMs) or binning with a fixed resolution, as measured by Wasserstein distance between samples.

trained with expectation-maximization, and logistic regression over a flat binning of the data (with a tuned bin size). Each model is trained with only 200 samples from the distribution. The results show that the decimal autoregressive language model (“Decimal AR”) performs extremely well, handling asymmetric, multimodal, and heavy-tailed distributions, which are among the diverse types characteristic of time series data.

4.5 LLMTIME: RETROFITTING LLMs FOR FORECASTING

Forecasting with `LLMTime` has relatively few steps. Once the numerical values are processed into strings, making predictions with the language model follows standard sampling procedures. As we show next, however, correct pre-processing is not always intuitive but is extremely important, and incorrect handling can lead to unusable predictions.

TOKENIZATION Tokenization is particularly important because it directly influences how patterns form within tokenized sequences and the types of operations that language models

can learn. Unfortunately, common tokenization methods like BPE tend to break a single number into tokens that don't align with the digits, which can make arithmetic considerably more difficult [146]. For example, the number 42235630 gets tokenized as [422, 35, 630] by the GPT-3 tokenizer, and changes by even a single digit can result in an entirely different tokenization. By contrast, in many new open-source LLMs (e.g. LLaMA [208]), numbers are tokenized into individual digits by default. To remedy the tokenization of GPT models, we separate the digits with spaces to force a separate tokenization of each digit and use a comma (",") to separate each time step in a time series. Because decimal points are redundant given a fixed precision, we drop them in the encoding to save on context length. Thus, with e.g. 2 digits of precision, we pre-process a time series as follows before feeding into the tokenizer:

$$0.123, 1.23, 12.3, 123.0 \rightarrow " 1 2 , 1 2 3 , 1 2 3 0 , 1 2 3 0 0 " .$$

In [Figure 4.3](#), we show that the added spaces of this encoding are helpful for GPT models, preventing the model from getting derailed by outputting an unusual token during sampling. For LLaMA models, with their unique tokenization of numbers, added spaces have the opposite effect. Each digit and space is already assigned its own token, and space tokens become nuisance inputs, adding to the sequence length without simplifying the sequence's structure and potentially making the sequence out-of-distribution to the model.

RESCALING To avoid wasting tokens when the inputs are very large, we scale values down so that the α -percentile of rescaled time series values is 1. We avoid scaling by the maximum value so that the LLM can see some fraction of examples $(1 - \alpha)$ where the number of digits changes and reproduce this behavior in its outputs to produce larger values than it has seen. We also experiment with an offset β based calculate as a percentile of the input data, and we tune these two parameters on validation log likelihoods (details in [section B.1](#)).

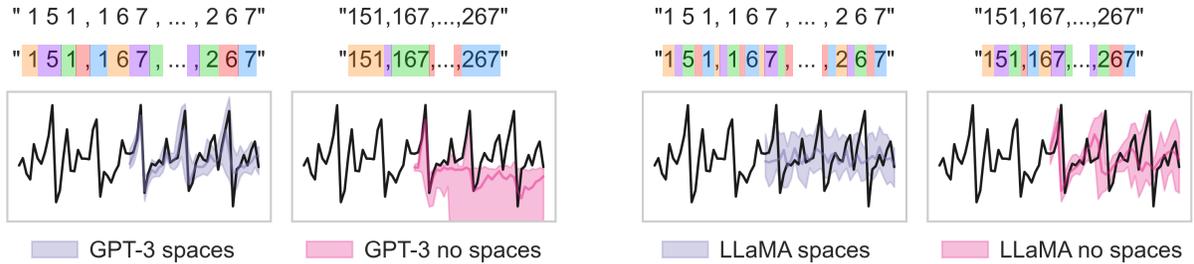


Figure 4.3: Careful tokenization is important for good forecasting with LLMs. Using the Australian Wine dataset from Darts [107], with values [151, 167, ..., 267], we show the tokenization used by GPT-3 [30] and LLaMA-2 [206] and the corresponding effect on forecasting performance. Added spaces allow GPT-3 to create one token per digit, leading to good performance. LLaMA-2, on the other hand, tokenizes digits individually, and adding spaces hurts performance.

SAMPLING / FORECASTING To forecast, draw many samples (e.g. 20) from the LLM and use the statistics of the samples at each time step to construct a point estimate (e.g. as the median) or probabilistic forecast (e.g. as quantiles). To control sampling, we use temperature scaling, logit bias, and nucleus sampling (Appendix B.2).

4.6 TIME SERIES EVALUATIONS

TIME SERIES DATA Time series data typically takes the exact same form as language modeling data, as a collection of sequences $\{U_i = (u_1, \dots, u_j, \dots, u_{n_i})\}$, but in time series u_j is numerical. Because language models are built to represent complex probability distributions over sequences, they are theoretically well-suited for time series modeling. In practice, however, language models are held back by the details of tokenizing numbers. BPE compresses numbers based on frequency of occurrence in the training data, so numbers can be broken down into awkward chunks that make learning basic numerical operations challenging. Touvron et al. [208] therefore designed the LLaMA tokenizer to map numbers to individual digits, which can lead to significant improvements in mathematical abilities, with small LLaMA models outperforming GPT-4 [146].

The other challenge of applying language models to time series data is proper evaluation.

Mean absolute error (MAE) is commonly used but ignores uncertainty in the forecast which is highly limiting for stochastic data [20, 108]. While demonstrating steady improvements, research on deep learning for point prediction frequently ignores a key but simple fact: the real world is complex and predicting the future accurately from past observations alone is often impossible. In highly structured time series, such as observed traffic, with strong periodicity from both daily and weekly cycles, we may be able to forecast point predictions with a high degree of accuracy. However, in *stochastic* time series which display only modest structure (e.g. periodicity or seasonality), such as precipitation or wind speed patterns, we cannot hope to produce accurate predictions of specific future outcomes using only historic observations.

Researchers working in architecture design for timeseries frequently overlook intrinsic stochasticity in benchmark datasets. In Figure 4.4 we show how even sophisticated methods struggle to forecast accurately on data with a low signal-to-noise ratio. In Table 4.1 we take this observation further and show that, shockingly, naive constant predictors outperform two state-of-the-art time series models [130, 254] on several widely reported MSE evaluations. The numbers shown are taken directly from Challu et al. [33], Wu et al. [230], and Zhou et al. [254], but include new columns showing the performance of simply predicting either the mean or the last value of the observations. Notably, these datasets span many domains of practical interest (finance, energy, and climatology), and contain varying levels of structure and periodicity.

Continuous ranked probability score (CRPS) better captures distributional qualities and can compare models that generate samples without likelihoods. For a single prediction, the CRPS score is defined against the estimated cumulative distribution function (CDF), \hat{F} as $CRPS(\hat{F}, y) = \int_{\mathbb{R}} (\hat{F}(z) - \mathbb{I}_{(z-y)>0})^2 dz$, where $\hat{F}(z)$ is the empirical CDF produced by sampling forecasts and \mathbb{I} is the indicator function. While CRPS is an improvement over MAE, it also ignores key structures in the data, such as correlations between time steps.

Table 4.1: Multivariate results with varying prediction lengths. Bolded results indicate the best performing model, and italics the second best. In all cases simple statistics of the input data to the model are either the first or second best performing models in terms of both MSE and MAE accuracy. Historical Inertia (HI) [53] was also introduced as a trivial baseline but has worse performance than our constants.

Models	Mean		Last Value		N-HiTS		Autoformer		Informer		ARIMA		HI		
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	
Exchan.	96	0.139	0.269	0.081	0.196	<i>0.092</i>	<i>0.211</i>	0.197	0.323	0.847	0.752	0.296	0.214	0.112	0.251
	336	0.384	0.454	0.305	0.396	<i>0.371</i>	<i>0.443</i>	0.509	0.524	1.672	1.036	2.298	0.467	0.434	0.517
	720	0.938	0.736	0.823	0.681	<i>0.888</i>	<i>0.723</i>	1.447	0.941	2.478	1.310	20.666	0.864	0.955	0.772
ETTime2	96	0.150	<i>0.272</i>	0.203	0.312	0.176	0.255	0.255	0.339	0.365	0.4536	0.225	0.301	<i>0.158</i>	0.261
	336	0.205	0.313	<i>0.270</i>	<i>0.361</i>	0.295	0.346	0.339	0.372	1.363	0.887	0.370	0.386	0.379	0.438
	720	0.261	0.350	<i>0.335</i>	<i>0.401</i>	0.401	0.426	0.422	0.419	3.379	1.388	0.478	0.445	0.446	0.477
Weather	96	<i>0.216</i>	0.271	0.259	<i>0.254</i>	0.158	0.195	0.266	0.336	0.300	0.384	0.217	0.258	0.345	0.339
	336	<i>0.313</i>	<i>0.336</i>	0.377	0.338	0.274	0.300	0.359	0.395	0.578	0.523	0.330	0.347	0.529	0.440
	720	<i>0.380</i>	<i>0.377</i>	0.465	0.394	0.351	0.353	0.419	0.428	1.059	0.741	0.425	0.405	0.545	0.439

Fortunately, language models can assign likelihoods to full sequences of time series data, and we show how a small modification to an LLM’s discrete likelihood can yield a continuous density that is useful for model comparison.

4.7 LLMTIME EVALUATION

We evaluate the zero-shot forecasting ability of LLMs by comparing LLMTIME with GPT-3 and LLaMA-2 70B to many popular time series baselines on a variety of benchmark time series datasets. Not only is LLMTIME able to generate plausible completions of the real and synthetic time series, it achieves higher likelihoods and CRPS values in zero-shot evaluation than the dedicated time series models like ARIMA, TCNs, and N-HiTS. When evaluated on deterministic metrics like MAE, LLMs also perform well, obtain the best or second best MAE values on each benchmark. As we are using LLMs with undisclosed datasets, data leakage is an important concern that we address directly in [subsection 4.7.3](#). Beyond strong performance on standard benchmarks, which are the most useful for comparison, we find that LLMTIME also performs well on datasets that could not have been present in the base

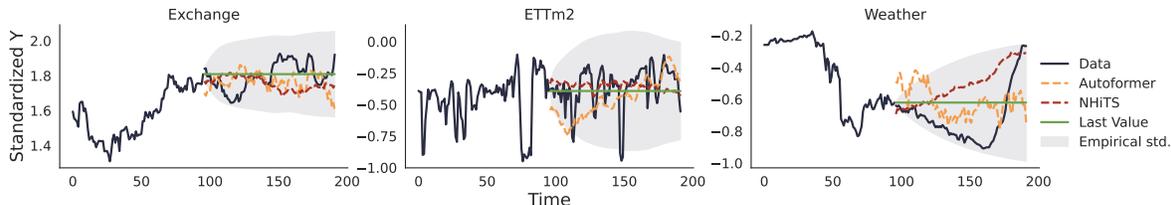


Figure 4.4: Example predictions on exchange rate (**left**), ETTm2 (a sequence of electricity transformer temperature readings, **center**), and weather (**right**) for NHiTS [33], Autoformer [230], and last value predictions, as well as the historical standard deviation of the change from the last observed value. On the exchange [130] and ETTm2 [254] datasets there is minimal structure to be exploited except on very short horizons, and forecasts tend to under-perform simple baselines. On semi-structured datasets like weather, models can capture some overall structure, such as NHiTS accurately predicting the final values in the forecasting window, but are still only on par with naive predictions. From these plots we see why probabilistic evaluation is necessary and point estimates are insufficient.

model’s training data. The full set of hyperparameters used for LLTime and the baseline methods are detailed in Appendix B.2.1. For some of the longer time series, not all of the history can be fit into the context window, and hence hyperparameters implicitly capture the trade-off between higher precision and capturing a larger temporal history.

DATASETS We use three benchmark datasets that are common within deep learning research and many baseline methods that accompany the benchmark datasets.

- **Darts** [107]: A collection of 8 real univariate time series datasets. For Darts, we use several methods that are implemented directly in the package, including neural network models (TCN [134], N-BEATS [170], N-HiTS [33]) and simple moving average models (ARIMA [24]). Darts enables learning observation models with tractable likelihoods and is therefore especially useful for benchmarking the probabilistic predictions of LLTime. We also include Spectral Mixture Gaussian Process (SM-GP) [226], a Bayesian nonparametric method (details in Appendix B.2.1).
- **Monash** [81]: The Monash forecasting archive contains 30 publicly available datasets along with baseline numbers for 12 forecasting models, including simple exponential smooth (e.g. ETS [117]), gradient boosting (e.g. CatBoost [175]) and deep learning

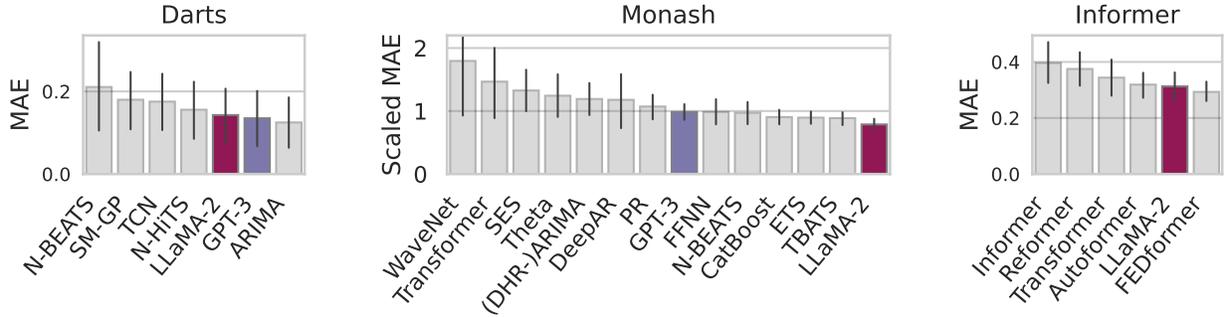


Figure 4.5: LLMTIME with base model GPT-3 or LLaMA-2 70B has the best or second best aggregated performance on several deterministic time series benchmarks [81, 107, 254] while being entirely zero-shot. Collectively, these benchmarks comprise 29 individual datasets with diverse sources, lengths, and noise levels. For Monash MAE numbers, established results are reported on unnormalized data, so we normalize values before aggregating (Appendix B.2.2). The Informer datasets are multivariate, and we predict each covariate independently with LLMTIME (Appendix B.2.3). GPT-3 evaluation on the Informer datasets was skipped because of the cost of API queries. Error bars show standard errors over the individual datasets in each benchmark.

models (e.g. DeepAR [184], WaveNet [168]). The Monash archive comprises over 400,000 individual time series, making it infeasible to use in its entirety with the largest available LLMs. To reduce the computational burden, we evaluate GPT-3’s zero-shot performance on 19 datasets described in Appendix B.2.2.

- **Informer** [254]: We evaluated on multivariate datasets widely used for benchmarking efficient transformer models [64, 254]. In order to predict multivariate data with LLMTIME, we forecast each covariate independently. We baseline against numbers obtained by running public implementations from the Autoformer [230] and FEDFormer [256] codebases (Appendix B.2.3).

4.7.1 DETERMINISTIC FORECASTS

To compute MAE values for LLMTIME we use the pointwise median of 20 samples from the base model (GPT-2 or LLaMA-2 70B). Figure 4.5 shows that deterministic predictions from LLMTIME are ranked best or second best on all the considered benchmarks while having no

trainable parameters. We provide visualizations of the forecasts in Appendix B.2.5, B.2.7, and B.2.8.

4.7.2 PROBABILISTIC FORECASTS

In Figure 4.6, we show several probabilistic evaluations on the Darts datasets, including aggregated NLL and CRPS numbers, as well as analysis of how each model reacts to decreasing the input data size. Evaluated on log likelihood and CRPS, `LLMTime` considerably outperforms the baselines in aggregate and on almost every individual dataset (results per dataset included in Appendix B.2.5). Given our analysis of language model-derived densities, it is unsurprising that language models excel in probabilistic evaluations, outperforming the baselines even more dramatically. In Figure 4.6 (left) we show two informative examples that capture the performance of `LLMTime`. When extrapolating the `AirPassengers` dataset, `LLMTime` successfully identifies and continues trend and period components, with uncertainty that grows as predictions get further from the input data. On `GasRateCO2`, `LLMTime` replicates local structure when there is relatively little global structure. In Figure 4.6 (right) we show that `LLMTime` not only performs better than baselines with access to the full training data but also when restricted to small fractions of the training data. As time series is frequently characterized by relative data scarcity and challenges in transfer learning, the data efficiency of LLMs is especially attractive.

COMPARISON WITH PROMPTCAST Though included in the results described above, we want to explicitly highlight that `LLMTime` significantly outperforms `PromptCast` [234] when applied to both GPT-3 and LLaMA-2 70B, according to CRPS and MAE aggregated over the Darts datasets. This performance gap highlights important differences between the two approaches. Unlike our method, `PromptCast` formulates forecasting as a conventional question-answering task in NLP by prompting pre-trained language models with an explicit

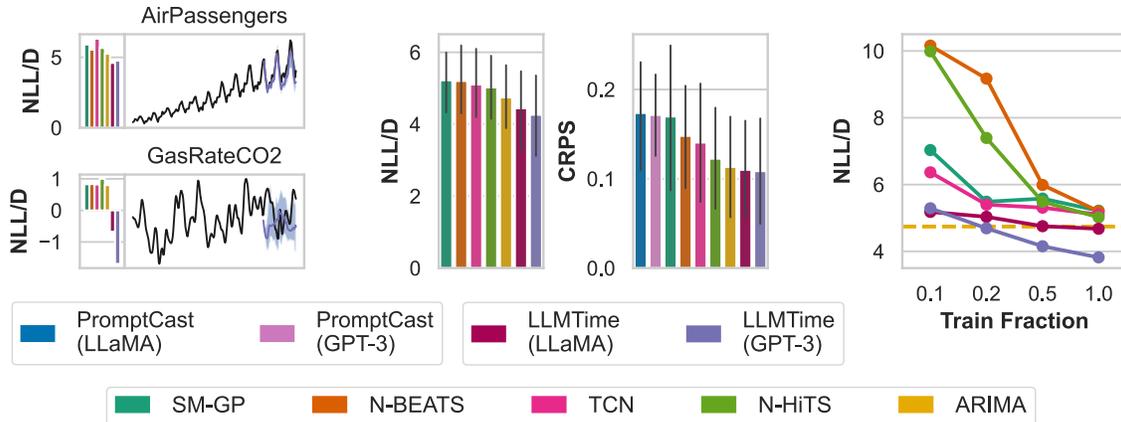


Figure 4.6: Extended experiments on the Darts datasets. **Left:** Example probabilistic forecasts with baseline negative log likelihood per dimension (NLL/D). LLMs easily extrapolate trends (e.g. AirPassengers) and reproduce local patterns when data is noisy (e.g. GasRateCO2). **Center:** When using probabilistic metrics like NLL and CRPS, LLMTIME outperforms all baselines, including PromptCast [234], a competing LLM method. Error bars show standard errors over datasets with Darts. **Right:** LLMTIME is much more sample efficient than competing methods. While the performance of other methods degrades rapidly when we restrict them to a fraction of the original training set, LLMTIME can assign high likelihood with only a few examples.

question about future values in a time series. For example, PromptCast feeds in the prompt “The values in the WoolyDataset for the past 95 time steps are 6172, 6709, 6633, . . . , 6077. What will the values for the next 5 time steps be? The values for the next 5 time steps will be”, to extract predictions from an LLM. PromptCast also does not apply our tokenization and data rescaling strategy, which we show is crucial for good performance.

4.7.3 TEMPORAL HOLDOUTS

Evaluating the performance of black box APIs, like those provided by OpenAI, can be challenging when training data for the underlying models is unknown. In our time series setting, it is natural to wonder if the common benchmark datasets we use are included in the GPT-3 training data. LLMs are known to memorize large amounts of their training data verbatim, including common benchmark text datasets and copyrighted material [21, 34]. Beyond outright memorization, more benign data leakage of closely related data is also

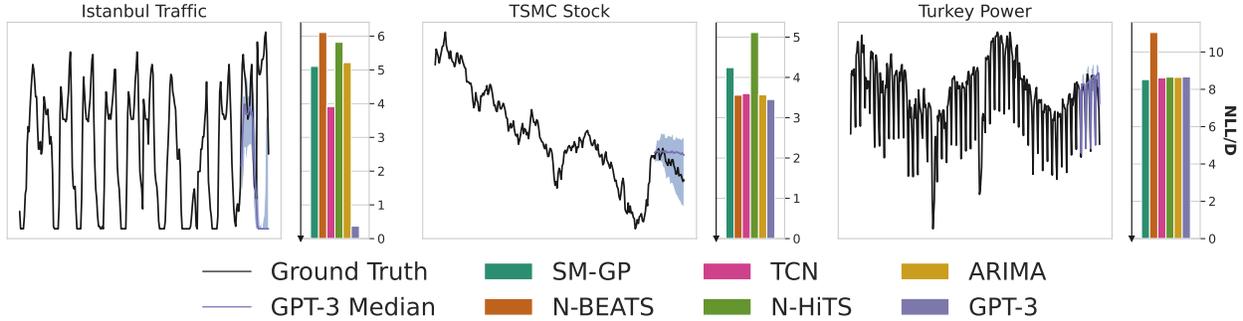


Figure 4.7: Evaluation on a collection of short univariate time series recorded after GPT-3’s training cutoff date. We compare the performance of our GPT-3 predictor against popular time series models. Predicted median and 10-90th percentile intervals are shown for GPT-3 given the context, and we compare test negative log likelihoods. GPT-3 continues to be competitive with or outperforms the baselines on all of the tasks, from in-context learning alone. This result reinforces our belief that GPT-3’s performance is not due to memorization of the test data.

possible, leading to overestimation of the generalization performance.

Even if our evaluation datasets are present in the GPT-3 training data, it’s unlikely that GPT-3’s good performance is the result of memorization for at least two reasons a priori. First of all, our idiosyncratic formatting is unlikely to be present in the training dataset, even if the numerical values and their order are the same. Second, the time series datasets are unlikely to appear in GPT-3’s training data sufficiently frequently to lead to memorization, as memorization increases in proportion with redundancy [137].

To further address the memorization concern, we also perform a direct experiment to show GPT-3 also demonstrates strong performance when evaluated on time series recorded after its training data cutoff date, September 2021. We use the following 3 time series:

- Istanbul Traffic (source: <https://www.kaggle.com/datasets/leonardo00/istanbul-traffic-index>): This dataset provides minute-by-minute Traffic Index data for Istanbul from October 2022 to May 2023. We select the "TI" column and downsample the series to an hourly frequency for the period from May 5th, 2023 to May 18th, 2023, resulting in a total of 267 observations.
- TSMC Stock (source: <https://www.kaggle.com/datasets/yeemeitsang/tsmc-sto>

[ck-exchange-2022](#)): This dataset contains daily stock market trading data for Taiwan Semiconductor Manufacturing Company Limited for the year 2022. We use the closing price column, which consists of a total of 246 observations.

- Turkey Power (source: <https://www.kaggle.com/dharanikra/electrical-power-demand-in-turkey>): This dataset includes hourly electricity generation and consumption data for Turkey from January 1st, 2020 to December 31st, 2022. We choose the "Total" column and downsample to daily data for the year 2022, resulting in 366 observations.

For each time series, we reserve the last 30 observations as test data and perform hyperparameter tuning for each method over the same grid as in Appendix B.2.1. As displayed in Figure 4.7, GPT-3 not only predicts plausible continuations of each time series but also competes with or even surpasses the performance of the baseline models in all the tasks, solely based on in-context learning. This result reinforces our belief that GPT-3’s performance is not due to memorization of the test data.

4.8 CONNECTIONS WITH SIMPLICITY BIAS

To understand why LLMs can extrapolate time series in a zero-shot manner, let’s take a step back and consider simple numerical sequences, for example $[1, 4, 9, 16, \dots]$ or $[0, 0, 1, 0, 1, 2, 0, 1, 2, 3, \dots]$. For any input sequence, there are arbitrarily many generation rules that are consistent with the input (e.g. $f(x) = x^2$ for $x \in [1, 2, 3, 4, \dots]$), but some generation rules are overly complex and will generalize poorly. LLMs can forecast effectively because they prefer completions derived from simple rules, adopting a form of Occam’s razor prior [58, 82, 196]. To explicitly demonstrate this phenomenon, we create a synthetic example using the function $f(x) = x + \cos(x)$ with additive Gaussian noise. We fit symbolic expressions to the first 70% of timesteps using PySR [50] with symbols $["+", "\cdot", "-", "/"]$,

“sin”, “cos”, “exp”, “square”] to identify generating rules with known complexity, quantified by the number of symbols in the regressed expression (Appendix B.3). Figure 4.8 (left) shows the likelihood that GPT-3 assigns the highest likelihood to symbolic regression generating rules that balance consistency with complexity.

In Figure 4.8 (right) we show how program induction in LLMs leads to good zero-shot prediction for many deterministic patterns common in time series data. Along with samples, we also show likelihoods, comparing against standard time series models, which often struggle to extrapolate these simple patterns because they cannot identify a programmatic generation rule to make predictions unlike those seen in the observed history. While the generic simplicity bias is helpful for identifying and extrapolating patterns in the input, a number of patterns common in time series models also translate directly to known capabilities of language models, for example

- **Repetition bias and periodicity:** LLMs’ bias towards repetitive sequences [111] (often unwanted in NLP) corresponds precisely to the ability to identify and extrapolate periodic structure in the input. 4.2, 8.6, 1.0, 4.2, 8.6 will lead to a 1.0 as a likely next output without any time series or arithmetic knowledge ($x_t = x_{t-T}$).
- **Arithmetic and trend components:** LLMs’ ability to perform addition and multiplication [146, 239] maps on to extrapolating linear and exponential trends. For example, predicting the next element of 0.2, 1.6, 3., 4.4 the LLM needs only to add 1.4 to the last element ($x_{t+1} = x_t + c$). Similarly, exponential trends have the generating rule $x_{t+1} = c \cdot x_t$ and sigmoid trends have the generating rule $x_{t+1} = x_t + cx_t(1 - x_t)$.

Combining multiple patterns together presents a more difficult challenge, as it requires both identifying the composite pattern and being able to perform the multiple operations within the same token budget. Supposing that a model can perform copying in a single forward pass and addition in a single forward pass, that does not necessarily mean that it can do both

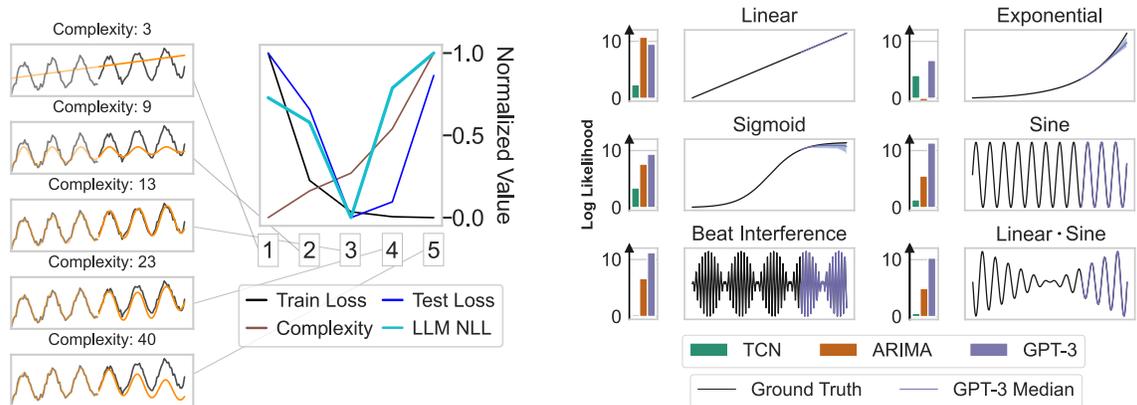


Figure 4.8: LLMs can find low complexity explanations of the data, enabling them to zero-shot extrapolate numerical sequences. **Left:** GPT-3 likelihoods favor solutions from symbolic regression (PySR [50]) that balance training loss and complexity, leading to good generalization. **Right:** GPT-3 predicted median and 10-90th percentile prediction interval are shown given 140 timesteps of context. On the right of each time series, we show the log likelihoods compared to the ARIMA and TCN time series models. Overall, GPT-3 performs considerably better than the baselines, though composition and exponential growth are more challenging for the models (Appendix B.3.1).

simultaneously. We find that GPT-3 is only sometimes able to perform these compositions, though GPT-4 does so more consistently as shown in section B.4. It is likely that the limitations on compute and tokens spent may make this composition unnecessarily hard, and that additional recursive structure, for example from a scratchpad [164], Chain of Thought (CoT) prompting [220], or adaptive computation [7, 189], would make this task easier.

4.9 PROMISES OF JOINTLY MODELING TEXT AND NUMERICAL DATA

So far we’ve shown that LLMs are effective forecasters across a variety of datasets and that their forecasting ability arises from biases created by generative pretraining. **LLMTime** offers a mechanism for large-scale pre-training that is uncommon in machine learning for time series. LLMs lessen the amount of time series data that must be aggregated for pretraining, substituting text pretraining in its place, and enable more powerful scaling results. Beyond

escaping the limits of task-specific data, text pretraining also has many test-time benefits that stem from the base model’s ability to process and generate natural language. As we show in the following section, LLMs can leverage their abilities in order to seamlessly incorporate missing data or answer questions about time series.

BASE MODELS AND FORECASTING PERFORMANCE Given the rapid growth and improvement in open-source LLMs [206, 208], the relationship between LLTime forecasting performance and the performance of the underlying base model is particularly important and has broad implications. Steady increases in LLM benchmark performance can directly translate to steady improvements in forecasting ability. In Figure 4.9 (right), we show a study with OpenAI models (davinci, babbage, curie, ada), variants of LLaMA [208] (7B, 13B, 33B, 65B) and LLaMA-2 [206] models (7B, 13B, 70B) measuring accuracy on the Massive Multitask Language Understanding (MMLU) benchmark and probabilistic forecasting error. As we might hope, when reasoning (MMLU) performance increases forecasts also improve.

CHAT MODELS Though convenient scaling relationships appear to hold for base models, the correlation begins to break down when we consider models that have been post-processed for chatbot applications. GPT-4 [169], for example, demonstrates considerably more intelligence than GPT-3 and LLaMA models in natural language tasks, but effectively applying it to time series is challenging. In Figure 4.9 (center), we show that GPT-4 has a forecasting error (CRPS) significantly larger than GPT-3’s on the Darts datasets. The performance drop is the result of several small details in GPT-4’s method. Due to the altered tokenization, GPT-4 cannot be easily forced to tokenize individual digits into an unbroken stream of numbers. Due to the restrictions on the API, likelihood evaluations are also not permitted, which is why we present results for only CRPS. While GPT-4 can perform well on synthetic examples (shown in section B.4), we find that the GPT-4 calibration is much worse than GPT-3’s on stochastic data, likely as a result of the preprocessing details above and the fact that the

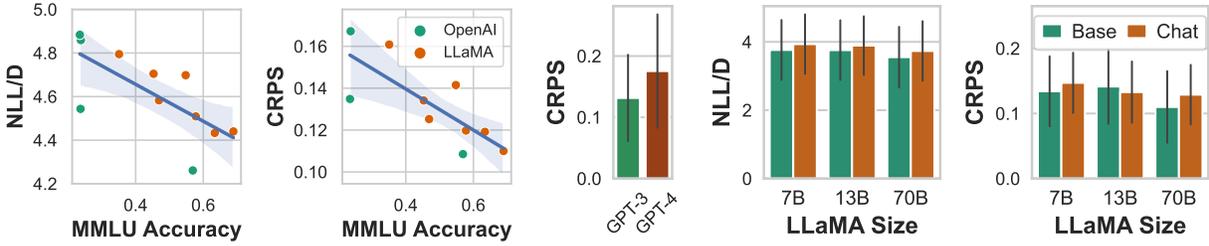


Figure 4.9: Left: Time series forecasting performance (NLL/D and CRPS on Darts [107]) improves with reasoning performance of the underlying model LLM, as judged by accuracy on the Massive Multitask Language Understanding (MMLU) benchmark [105]. Displayed results are for all GPT-3, LLaMA [208], and LLaMA-2 [206] base models. **Center:** GPT-4 performs worse than GPT-3. **Right:** Forecasting performance (NLL/D and CRPS on Darts) appears to be negatively affected by alignment procedures (e.g. instruction tuning and RLHF) in general. LLaMA-2 chat models typically perform worse than the corresponding base model. Error bars show standard errors over individual datasets.

model has been treated with RLHF [46] which is known to degrade calibration on question-answering tasks [169]. GPT-4 is not the only example of degraded performance in models designed for chat functionality. We observed the same phenomenon in LLaMA-2 models, which have corresponding chat versions for every model size. Figure 4.9 (right) shows that chat versions tend to have markedly worse forecasting error than their non-chat counterparts, though still maintain trends in size and reasoning ability.

MISSING DATA A major advantage of treating forecasting as a text completion task and using LLMs is that we can easily feed in any input that can be encoded as text. Often in time series, the time series will be incomplete and certain observations are missing. Simple imputation methods, such as nearest neighbor, are still core pre-processing steps in common data science workflows [161], and the choice of imputation method is especially relevant to clinical data, which often contains irregularly sampled measurements and where missingness can be meaningful signal in itself [128]. Much like humans reading partial reports, LLMs can handle missing values without imputation by adopting special symbols, for instance,

[64, , , 49, , 16,] \rightarrow "64, NaN, NaN, 49, NaN, 16, NaN"

. In [Figure 4.10](#) we compare likelihoods and CRPS value for forecasts from traditional time series methods and LLaMA-2 70B on data that has been corrupted with missing values and then processed with linear interpolation and the above string formatting. While the likelihoods of traditional methods rapidly deteriorate with corruptions, we find that LLaMA-2 70B is more resilient, and when comparing CRPS values, LLaMA-2 70B is competitive with methods that use interpolation.

CONNECTING TIME SERIES AND TEXTUAL UNDERSTANDING Because LLMs are designed for natural language and code, we can augment the numerical time series with useful text. We can do so either by providing textual side information as inputs, or by producing textual outputs from a given time series. An interesting question is whether GPT-4 can explain in text its understanding of a given time series. We probe this quality by providing GPT-4 the code to generate our synthetic time series, provide the values of one these time series, and then ask it to infer which of the functions produced the data in a zero-shot manner. The prediction accuracies are shown in [Figure 4.10](#), with the three remaining rows all being 0. With CoT [\[220\]](#) prompting the model performs much better than random chance; however, its ability to identify patterns better when directly extrapolating the numerical data, suggesting that its numerical understanding is not fully connected to its textual understanding. In making predictions, the model often explains properties of the time series in order to select the right candidate from the list, and we show several of these sample explanations in [section B.5](#). We also show how this task is encapsulated in a simple (unprompted) next token prediction problem on cells of a Jupyter notebook, illustrating why we expect such capabilities to emerge with a sufficiently powerful language model.

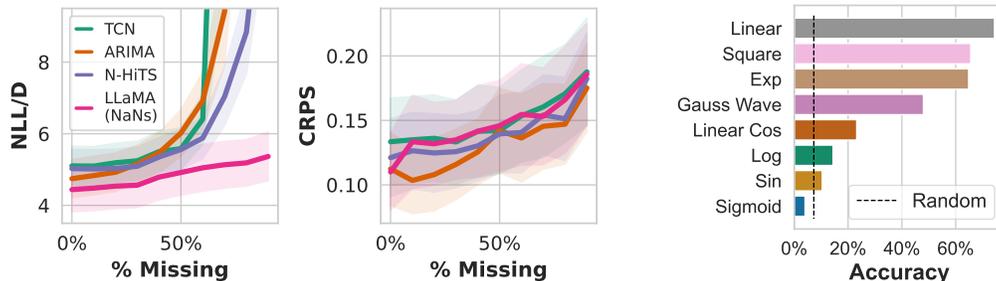


Figure 4.10: Left: LLMTime can handle missing values without interpolation by denoting missingness with text (e.g. ‘NaN’). For baseline methods we perform linear interpolation and then fit the model as usual. LLMTime assigns higher log likelihood to datasets preprocessed with added ‘NaN’s than baseline methods assign to interpolated datasets. Forecasting performance, as judged by CRPS, is competitive between LLMTime and alternative methods that use explicit interpolation. Filled area shows standard error over individual datasets and 3 random seeds. **Right:** LLMs can be used to answer questions about time series data posed as text. We show GPT-4’s accuracy at predicting the function that generated the time series, obtained using chain-of-thought prompting.

4.10 HETEROGENEOUS SCIENTIFIC DATA

Scientific data from chemistry or biology experiments is inherently multimodal, comprising a rich tapestry of heterogeneous data types. For instance, experimental datasets may include categorical features—such as sample labels, experimental conditions, or molecular identifiers—as well as numerical measurements like concentration values, reaction rates, or gene expression levels. In addition, these data often incorporate unstructured text descriptions (e.g., experimental protocols or expert observations) and may even include images or spectroscopic profiles that provide further context. This blend of discrete and continuous modalities presents unique challenges for deep generative models, which must learn to capture complex interdependencies and jointly represent diverse data distributions.

Language models, however, offer a sensible unifying paradigm for scientific data by harnessing the universality and expressiveness of text. By converting all modalities into a text-based representation—translating numerical data into descriptive language, summarizing categorical labels, or captioning images and spectra with rich contextual details—we can

consolidate disparate data types into a common format. This unified textual representation leverages the remarkable ability of language models to understand context, semantics, and intricate relationships, thereby simplifying integration, analysis, and generation of scientific data. In effect, language models not only overcome the hurdles posed by multimodal data but also provide a flexible and interpretable framework that aligns well with the diverse nature of scientific inquiry.

4.11 CONCLUSION

We have demonstrated that large language models can be used as pretrained time series forecasters by encoding numerical values as text. As with other “foundation” models, pretraining confers useful biases toward generalizable patterns that would otherwise be engineered into the model through architecture design [90], and enables natural scaling of performance with improvements in the base pretrained model. Because LLM forecasters are trained on language, they also confer unconventional capabilities, such as question answering. More broadly, framing time series forecasting as natural language generation can be seen as another step towards unifying more capabilities within a single large and powerful model, in which understanding can be shared between many tasks and modalities. Moreover, *zero-shot* forecasting can enable broadly compelling performance without requiring significant computational resources, domain expertise, or many downstream training data points.

While LLM forecasters benefit from the strengths of pretrained transformers, they also inherit their weaknesses, which can include a limited context window. While many univariate time series problems can fit comfortably within increasingly large context windows, multivariate problems pose a more significant challenge. There have been several recent advances extending LLM context windows to 10-100K tokens [2, 8, 10, 169]. Combining these advances with time series forecasting is a particularly exciting direction for future research.

Another potential challenge of using current LLMs architectures could be their weakness in arithmetic and performing recursive and compositional operations, which could be a limitation on particularly challenging time series. On the other hand, many time series do not require precise arithmetic. Understanding the extent to which this is the case, and relaxing this limitation, is also a promising avenue for future research. Separately from any limitation, it would also be promising to investigate effective procedures for fine-tuning LLMs on time series. We hope that bridging LLM research with time series forecasting brings benefits to both communities.

5 | FINE-TUNED LARGE LANGUAGE MODELS GENERATE STABLE INORGANIC MATERIALS AS TEXT

5.1 INTRODUCTION

Large language models (LLMs) are trained to compress large text datasets, but can also act as strong foundations for non-text data [58]. As compressors, LLMs extract common patterns and find simple programs that can produce them [82, 196], regardless of the data’s origin. From text pretraining alone, LLMs can compress or extrapolate data as diverse as images [58], tabular data [82], time series [87], or robotic trajectories [159]. Alongside generality, LLM pre-training also gives rise to sample efficiency, as in-context learning and fine-tuning require far fewer training examples to identify salient patterns than training a model from scratch [30].

The generality and sample efficiency of LLMs make them particularly promising for scientific problems, where data are often limited, collected from diverse sources, or challenging for non-experts to interpret. In materials science, for example, the number of known stable materials is relatively small, and the data describing each material are diverse, including composition, structure, and complex properties. LLMs can learn generalizable rules from a

small number of examples [258], combine modalities into a single model [160], and provide users with a text-based interface. A text interface, in particular, has the potential to improve access to scientific discovery [223]; LLMs can use text to describe new observations, or, in design applications (e.g. materials design, drug discovery), LLMs can ingest text that specifies desired properties or constraints [25].

In this work, we show that fine-tuned LLMs can generate the three-dimensional structure of stable crystals as text (Figure 5.1). Our method is simple: first, encode crystals as new-line separated strings and combine with text instructions, then perform parameter efficient fine tuning (PEFT) on a base LLM (LLaMA-2) with a multitask curriculum and translation augmentations. We evaluate our method with Materials Project data [118], comparing against an invariant diffusion model and a sequence model trained from scratch. Using both learned ML potentials and gold-standard DFT calculations, we show that our method can generate materials predicted to be stable at higher rates than baseline methods. To understand the success of our fine-tuning approach, we probe the learned symmetry properties of our model, proposing a new metric for language models trained on atomistic data and examining the effect of model scale on learned invariance. Going beyond unconditional generation, we also show that our LLMs have other useful abilities within materials design, such as text-conditional generation and infilling, which can be used to optimize the properties of existing materials.¹

5.2 RELATED WORK

There are two central challenges in applying generative models to crystals and related atomistic data. The first challenge is that atoms are intrinsically both discrete and continuous objects, as each atom has both an element identity and a position in three dimensional

¹<https://github.com/facebookresearch/crystal-llm>

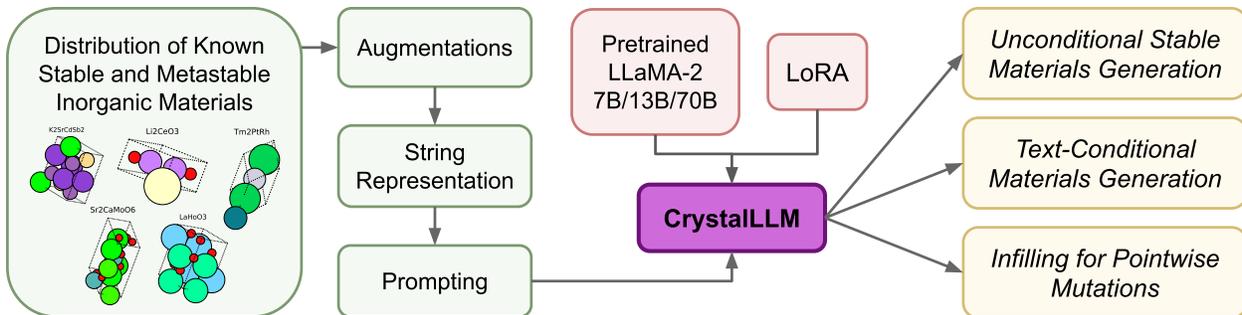


Figure 5.1: Overview of our approach to materials generation with large language models. Using string formatted crystals and task-specific prompting, we enable unconditional stable materials generation, text-condition materials generation, and structural infilling. Base LLaMA-2 models are fine-tuned on a database of known inorganic materials [147] using low-rank adapters.

space. Approaches to generative modeling often differ between for discrete and continuous data, and modeling both simultaneously can be significantly more complex than modeling either individually. The second key challenge is the prevalence of symmetries in atomistic data. The unit cell, a repeated pattern tiled infinitely in every direction, is the common representation for crystals because it easily captures translation invariance, the fact that atoms can be shifted and wrapped around the unit cell while still representing the same underlying structure. Symmetries can pose challenges to deep learning models because they entail constraints on the functions that neural networks can learn.

DIFFUSION MODELS Xie et al. [232] introduced crystal diffusion variational autoencoder (CDVAE) to directly deal with both of these challenges. CDVAE uses several individual generative models for discrete and continuous components that share a continuous (VAE) latent space. The chemical composition is reconstructed from this latent space using a language modeling head, while atom positions are generated with a denoising diffusion model [109]. Since CDVAE, several works have extended diffusion processes to capture all parameters of the crystal, not just the atomic coordinates. Both Jiao et al. [120] and Zeni et al. [242] accomplish this by creating diffusions for the lattice parameters and atom identities, while Yang et al. [236] design a new continuous representation that unifies atom identities

and positions in a single high-dimensional tensor. In most cases, these diffusion models were designed with a careful eye towards symmetries and are built on top of graph neural networks with strict invariance/equivariance properties [120, 232, 242]. The approach of Yang et al. [236] is more similar to ours, as they apply a general-purpose architecture (3D U-net) and modeling approach (Gaussian diffusion) to a new representation, without guaranteeing symmetries. Discrete atom identities and variable length (number of atoms), however, require special considerations in diffusion models, unlike standard language models, which were originally designed for modeling discrete sequences.

LANGUAGE MODELS Flam-Shepherd and Aspuru-Guzik [72] demonstrate an alternative to continuous denoising models and architectural invariances. Instead of treating discrete and continuous modalities separately, as in CDVAE, Flam-Shepherd and Aspuru-Guzik [72] uses sequences of discrete tokens to represent everything, including the digits of atomic coordinates. With all data encoded as tokens, standard language modeling methods designed for text can be applied with little to no modification. The simplicity of this method also makes it simple to adapt to many different kinds of molecular structures, including small molecules, protein binding pockets, and, of course, crystals. In lieu of architectural symmetries, augmentations of the training data are used to encourage learning known invariances. Flam-Shepherd and Aspuru-Guzik [72] demonstrates that language models trained from scratch on many common molecular datasets actually outperform popular domain-specific models, including CDVAE, in their ability to capture valid element compositions and high-level statistics of the training data. Similarly, Antunes, Butler, and Grau-Crespo [11] also use language models to generate crystal structures as discrete sequences by training from scratch on millions of CIF strings.

OUR WORK In this work, we show that pretrained LLMs are also useful for understanding and generating 3-dimensional atomic structures. By using a pre-trained LLM, we can

achieve high rates of validity without crystal-specific tokenization [72] or millions of auxiliary structures [11]. Unlike many methods designed specifically for crystal structures and symmetries, our method can also be easily extended to multiple crystal generation tasks and, in the future, to other atomistic modalities without any changes to the underlying model or training procedure. Building on the basic observations made by Flam-Shepherd and Aspuru-Guzik [72], we show that larger models, which are often more effective compressors of data, demonstrate improved ability to learn symmetries from the training data and augmentation.

5.3 BACKGROUND

LANGUAGE MODELING LLMs perform next-token prediction over sequences. The model is a categorical distribution, $p(w_{t+1}|w_{0:t})$, where $w_{0:t}$ is the prompt, a sequence of input tokens, and w_{t+1} is the predicted next token. To generate sequences from the model, the conditional distribution is sampled sequentially, but samples are rarely drawn from the original, unmodified categorical distributions. Instead the sampling procedure is typically modulated with *temperature* (τ) and *nucleus size* (p) hyperparameters. Temperature serves to flatten the conditional distributions to uniform (high temperature) or collapse them around their maximal probabilities (low temperature). Nucleus size limits which tokens can be sampled based on the cumulative distribution function, clipping out values that contribute very little mass. A nucleus of p ($0 < p \leq 1$) corresponds to keeping tokens to cumulatively contribute $p\%$ of the total probability, and discarding the rest.

TOKENIZATION To train language models on text datasets, strings are converted into sequences of tokens. Most modern LLMs rely on *byte pair encoding* (BPE) [76], a compression method that assigns tokens to common substrings, making overall sequence lengths shorter. One downside of BPE tokenization is the default tokenization of numbers. BPE typically

breaks numbers into irregular substrings instead of individual digits. While breaking numbers into multi-digit tokens creates shorter sequences, it also complicates learning basic arithmetic operations, which typically operate at the level of individual digits. Luckily, Touvron et al. [206] introduce tokenizers for LLaMA-2 models that break numbers into a sequence of digits, which has been shown to dramatically improve performance on arithmetic tasks [146]. We use LLaMA models in our work because they have a natural representation of 3D coordinates and can therefore learn simple functions over those coordinates that obey domain-specific symmetries.

5.4 PARAMETERIZING BULK MATERIALS

Periodic materials are defined by a unit cell repeated infinitely along all three dimensions (Figure 5.2). The unit cell comprises a lattice (parallelepiped) with side lengths (l_1, l_2, l_3) and angles $(\theta_1, \theta_2, \theta_3)$. Within the lattice, there are N atoms, each specified by an element identity, e_i , and set of 3d coordinates (x_i, y_i, z_i) which can be absolute or fractional (specified as a percentage of the unit cell side lengths). Therefore a bulk material can be fully described by the tuple

$$C = (l_1, l_2, l_3, \theta_1, \theta_2, \theta_3, e_1, x_1, y_1, z_1, \dots, e_N, x_N, y_N, z_N) . \quad (5.1)$$

5.5 FINE-TUNING APPROACH

5.5.1 DATASET AND AUGMENTATIONS

DATASETS For consistency with prior work [73, 232] we used MP-20 [118], a dataset of 45231 materials, when training for unconditional generation. All structures in MP-20 are stable, and therefore an effective generative model trained on MP-20 should tend to propose

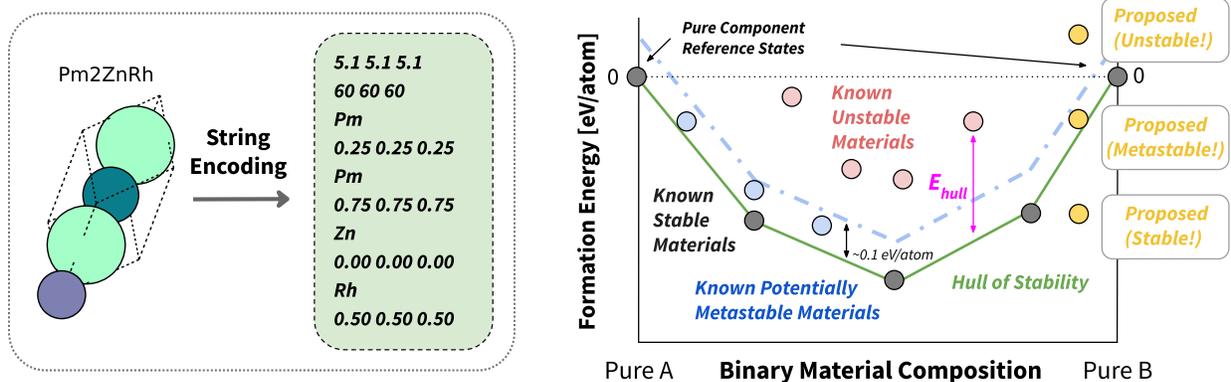


Figure 5.2: **(left)** We convert the crystal lattice, atom identities, and atom positions into strings. The model is trained to generate a structures conditioned on the text prompt, which might contain additional information about the composition, properties, or a starting structure to modify. **(right)** Energy above hull (E_{hull}) quantifies the stability of a material. A crystal with $E_{\text{hull}} < 0.1$ will be energetically favorable both in its structure and composition.

new crystals that are at least metastable. For text-conditioned generation, we train with all forms of prompting on a collection of 120,000 crystals from Materials Project (Appendix C.1.3). The collection includes basic property information, such as the space group number, band gap, E_{hull} and the chemical formula.

AUGMENTATIONS Crystals structures are symmetric under translational. All atomic coordinates can be shifted modulo the lattice boundaries without changing the resulting material structure. Similarly, the ordering of atoms within the lattice is irrelevant to the underlying material (permutation invariance). Prior work on diffusion generative models guarantee these symmetries as invariance or equivariance constraints on the model architecture [120, 232]. To encourage translation invariance in our language models, we apply random uniform translations to the fractional coordinates. We chose not to augment the ordering of atoms because these variables often contained valuable information, for example grouping set of elements together for placement in the lattice (discussion in Appendix C.1.1).

5.5.2 STRING ENCODING AND PROMPTS

Our approach to generating stable materials is pleasingly simple. We take a pre-trained LLM, which has useful biases towards generalizable patterns, and fine-tune it on crystal string representations. Because language models can also ingest text, we can condition the model’s generations on text descriptions. The flexibility of language models also allows us to solve other tasks, such as infilling, through small modifications to the input formatting. Though we focus solely on crystal structures in this work, our method itself is general purpose and could be easily extended to proteins, nucleic acids, or small molecules. We include a more detailed discussion of how general text-pretraining impacts our method in Appendix C.1.5.

STRING FORMATTING AND TOKENIZATION We convert the crystal tuple C (Equation 5.1) using fixed precision numbers. An example of crystal string formatting is shown in Figure 5.2. We represent lattice lengths with one decimal place (2-3 digits) and lattice angles as integers (1-3 digits). Fractional coordinates are always represented with two digits. 3D coordinates are combined with spaces and all other crystal components are combined with newlines. We deliberately chose LLaMA-2 models because they are both state-of-the-art in overall performance among open-source models and because they tokenize numbers as individual digits by default. Notably, it is therefore impossible to create one token per full number, as Flam-Shepherd and Aspuru-Guzik [72] do in their best performing model (further discussion in Appendix C.1.1). Instead, we rely on the extensive pretraining of LLaMA-2 models to instill useful biases over numerical operations [146].

PROMPT DESIGN To train a model that can be used for many tasks, including unconditional generation, text-conditional generation, and infilling, we use task-specific prompts. The input to the model is a prompt followed by the string-formatted crystal (Figure 5.2).

In the most basic case, the prompt indicates that the model should generate bulk materials represented as a lattice and atoms. The prompt can also be expanded to include a desired composition or material properties, or to include a starting structure, in the case of infilling. For infilling, the prompt includes the string-formatted crystal with every instance of a randomly chosen element replaced with [MASK], and the model is trained to generate the identity of the masked element at the end of the sequence. During training all three tasks are included through random sampling, with two thirds generation and one third infilling (details in Appendix C.1.2). As in instruction tuning, the prompt is given as input to the model but does not contribute to the generative loss function. The model is only penalized for its predictions on the crystal string or masked element.

Generation Prompt	Infill Prompt
<p><s>Below is a description of a bulk material. [The chemical formula is Pm2ZnRh].</p> <p>Generate a description of the lengths and angles of the lattice vectors and then the element type and coordinates for each atom within the lattice:</p> <p>[Crystal string]</s></p>	<p><s>Below is a partial description of a bulk material where one element has been replaced with the string “[MASK]”:</p> <p>[Crystal string with [MASK]s]</p> <p>Generate an element that could replace [MASK] in the bulk material:</p> <p>[Masked element]</s></p>

Blue text is optional and included to enable conditional generation. Purple text stands in for string encodings of atoms.

All of our experiments were conducted with LLaMA-2 models (7B 13B, and 70B) [206, 207] through the Transformers library [227] and PyTorch [174]. In order to train on small number of GPUs we use 4-bit quantization [61] and Low-Rank Adapters (LoRA) [112]. We provide the full hyperparameters and training details in Appendix C.1.4.

5.6 EVALUATING SAMPLES

For basic evaluation of the LLM samples, we use the validity and diversity metrics introduced by Xie et al. [232]. Structural validity is determined by non-overlapping atomic radii (overlapping taken to be both atoms within half a radius of each other), while compositional validity captures the net charge of the structure (only structures with net neutral total charge are valid). Diversity is computed as pairwise distance between samples under featurizations of the structure and composition from Matminer [219, 232].

While useful for sanity checking models, simple validity metrics only reflect a subset of our real-world priorities in generating novel materials. Arguably the most important property that we hope to assess in samples is their predicted stability, which we can approximate by predicting the energy of relaxed structures. Using known materials and energy calculations from Materials Project we construct the ground truth energy convex hull and then calculate the approximate energy above hull, \hat{E}_{hull} .

5.6.1 ENERGY ABOVE HULL

For a given set of environmental conditions, every crystal has a corresponding energy that describes how likely it will occur in a particular configuration. Configuration with unfavorable electrostatic interactions from unlike atomic positions, such as highly overlapping atoms, are typically high energy. The gold standard for energy prediction is density functional theory (DFT), which provides tractable approximations to governing quantum mechanical equations that describe the energy and time evolution of a system. DFT, however, can be prohibitively expensive, often scaling $O(n^3)$ with the system size, which has motivated development of deep learning potentials to approximate DFT solutions [132].

STABILITY OF HYPOTHETICAL MATERIALS (E_{hull}) The composition of a crystal also impacts its energy, as different elements have different geometries and charge properties. Certain stoichiometries, or ratios of elements, are naturally favored, and a composition of elements A and B with constituent parts A_xB_y can dissociate into the composition A_cB_d if it is energetically favorable. Because of the effect of composition, the energy of a crystal is typically a two dimensional concept captured by the energy hull, which is the minimum observed configuration energy for a given composition. For a crystal to be low-energy and stable, and therefore give rise to a practically useful material, it must have a small *energy above hull* (E_{hull}), the distance from the energy hull for the crystals elemental composition (Figure 5.2). Crystals with $E_{hull} < 0$ are considered stable and by definition have lower energy than the known minimum (which has $E_{hull} = 0$). Crystals with $E_{hull} < 0.1$ eV/atom are often *metastable* and likely to be practical useful [195].

We chose two methods to estimate material stability:

- **ML potential:** M3GNet [37] provides energy, force, and stress approximations for crystal unit cells. For each sample we first run a relaxation using force and stress approximations then use the energy of the final structure.
- **DFT:** We run a relaxation using the Density Functional Theory code VASP [95] with INCAR settings chosen by Pymatgen [167]. DFT is the more accurate, but also much more computationally intense, of the two options.

In both cases, results are compatible with Materials Project values [118] (Appendix C.2.1). Because DFT is prohibitively expensive for many use cases (often hours per calculation), we only use it to double-check results obtained with ML potentials, and we only run VASP calculations on materials that have already been predicted as metastable by M3GNet (< 0.1 eV/atom \hat{E}_{hull}). The use of a M3GNet surrogate model is not perfect as many structures in Figure 5.4 (right) have energies above the expected 0.1 eV/atom threshold, but the structures

Table 5.1: Following prior work [232], we evaluate fine-tuned LLaMA-2 models using validity, which captures physical constraints, as well as coverage and property metrics, which capture alignment between the ground truth and sampling distribution. We add stability checks, which count the percentage of samples estimated to be stable by M3GNet [37] and DFT [95] (details in Appendix C.2.2). LLaMA models generate a high percentage of both valid and stable materials.

Method	Validity Check		Coverage		Property Distribution		Metastable	Stable
	Structural \uparrow	Composition \uparrow	Recall \uparrow	Precision \uparrow	wdist (ρ) \downarrow	wdist (N_{el}) \downarrow	M3GNet \uparrow	DFT \dagger \uparrow
CDVAE	1.00	0.867	0.991	0.995	0.688	1.43	28.8%	5.4%
LM-CH	0.848	0.835	0.9925	0.9789	0.864	0.13	n/a	n/a
LM-AC	0.958	0.889	0.996	0.9855	0.696	0.09	n/a	n/a
LLaMA-2								
7B ($\tau=1.0$)	0.918	0.879	0.969	0.960	3.85	0.96	35.1%	6.7%
7B ($\tau=0.7$)	0.964	0.933	0.911	0.949	3.61	1.06	35.0%	6.2%
13B ($\tau=1.0$)	0.933	0.900	0.946	0.988	2.20	0.05	33.4%	8.7%
13B ($\tau=0.7$)	0.955	0.924	0.889	0.979	2.13	0.10	38.0%	14.4%
70B ($\tau=1.0$)	0.965	0.863	0.968	0.983	1.72	0.55	35.4%	10.0%
70B ($\tau=0.7$)	0.996	0.954	0.858	0.989	0.81	0.44	49.8%	10.6%

\dagger Fraction of structures that are first predicted by M3GNet to have $E_{\text{hull}}^{\text{M3GNet}} < 0.1$ eV/atom, and then verified with DFT to have $E_{\text{hull}}^{\text{DFT}} < 0.0$ eV/atom.

are largely close to the hull compared to the broader distribution of materials generated.

5.7 RESULTS

5.7.1 UNCONDITIONAL GENERATION

We sample 10,000 structures from each fine-tuned LLaMA model, parsing a CIF from the generated string. We reject the sample and draw another if a CIF cannot be parsed from the sampled string, which guarantees all samples can be interpreted as crystals but does not guarantee validity of the resulting crystal. We show the validity and predicted stability [232] of the resulting structures in Table 5.1, which shows that LLMs can achieve near-perfect rates of structural and compositional validity. Hyper-parameters like temperature and nucleus size can be used to trade-off validity and stability of samples

6.7 6.7 6.7
138.0 138.0 59.0
Ln
0.88 0.05 0.96
Ln
0.63 0.30 0.46
ln
0.38 0.55 0.96
ln
0.13 0.80 0.46

Figure 5.3: A sample with “hallucinated” element identities (Ln).

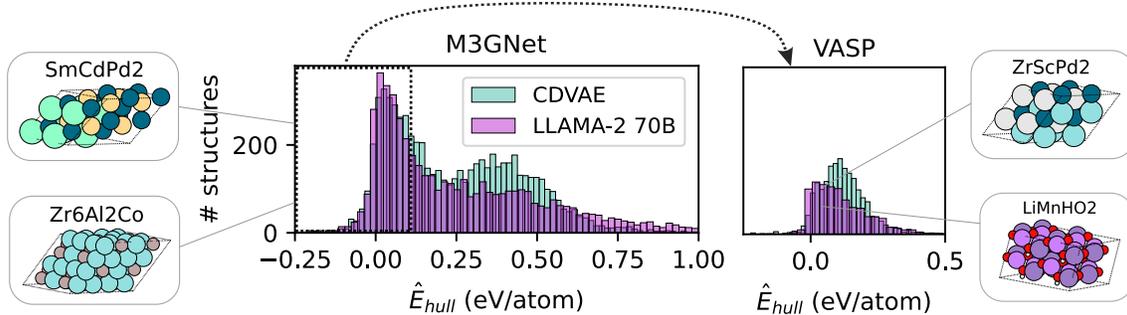


Figure 5.4: Stability of LLaMA samples compared to CDVAE [232]. Fine-tuned LLaMA-2 70B generates a higher rate of metastable ($\hat{E}_{\text{hull}} < 0.1$) and stable materials than CDVAE, using estimates of \hat{E}_{hull} from both M3GNet [37] and VASP [95]. Because of computational cost, we only run VASP on structures predicted to be stable by M3GNet. Stable materials generated by LLaMA are also more diverse (as quantified by Matminer featurization [219]) than stable samples from CDVAE. We include sampled stable structures, shown as (2,2,2) supercells, which display a high-degree of regularity and understanding of three-dimensional space.

with their coverage (Appendix C.2.3). LLaMA-2 70B strikes an effective balance, generating high rates of stable materials with good coverage and diversity (Figure 5.4). By default, generation is completely unconstrained and therefore the model can hallucinate imaginary elements, for example “Ln,” a common abbreviation for Lanthanide (Figure 5.3), but the problem can be easily avoided by constraining the tokens for element identities [227].

DIVERSITY, NOVELTY, AND SAMPLING SPEED When using generative models to discover new stable materials, there are several properties beyond the rate of stability that are practically significant. Novel and diverse samples encourage sufficient exploration of unknown material space, and sampling speed dictates how expensive it is to search within that space. We compare these properties for LLaMA-2 models and CDVAE in Figure 5.5. To calculate diversity and novelty, we use the same featurizations as in Table 5.1, calculating pairwise distances for diversity and distance to the closest neighbor in the training set for novelty (details in Appendix C.2.6). All metrics are computed over crystals judged metastable by M3GNet, so that all novelty and diversity are relevant and not an artifact of invalid generations. LLaMA-2 samples match or exceed the diversity of CDVAE samples and also obtain

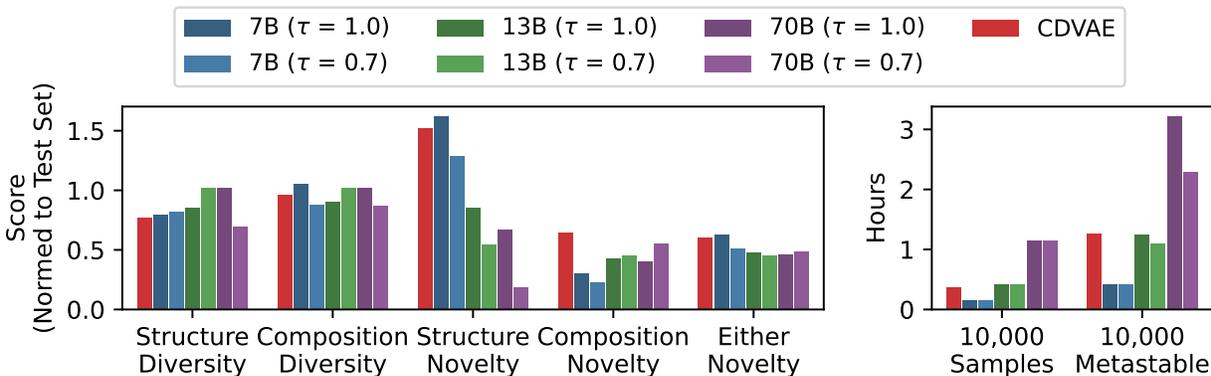


Figure 5.5: We compare LLaMA-2 models with CDVAE in their ability to generate novel and diverse samples as well as their overall speed. **(left)** We calculate diversity and novelty using a featurization of structure and composition (as in Table 5.1). Diversity is calculated as pairwise distance in feature space, while novelty quantifies the percentage of inputs that are far from the training set (Appendix C.2.6). All metrics are calculated only for samples that were already judged to be metastable. LLaMA-2 models often generate more diverse samples than CDVAE, and achieve similar overall rates of novelty. Interestingly, structural novelty is lower in larger models, while compositional novelty is higher. **(right)** We compare the time required to generate 10,000 samples from each model. We run LLaMA-2 models with the largest feasible batch size on one A100 GPU (Appendix C.2.7). While the largest LLaMA model is computationally expensive, smaller language models are very fast, especially when we consider both sampling speed and rate of stability.

high rates of novelty when we consider both composition and structure. Interestingly, larger LLaMA models display less novel structures but more novel compositions. It’s worth noting, however, that both CDVAE and LLaMA-2 7B far exceed the structural novelty of a held out test set, while 13B and 70B are just slightly lower. To judge sampling speed, we calculate the time required for 10,000 samples, using the largest possible batch size on one A100 GPU (Appendix C.2.7). In Figure 5.5, we compare the sampling speed with CDVAE and find that smaller models are often significantly faster when generating metastable samples.

5.7.2 CONDITIONAL SAMPLING

TEXT-CONDITIONED GENERATION Extending our method to text-conditional generation is as simple as including additional information in the prompt, with a small amount of additional text. We explore conditioning on spacegroup number, composition, and E_{hull} ,

as these properties are easy to verify (at least approximately) *in silico*. We assess the model’s ability to perform conditional generation by comparing the intended condition with labels obtained from an *in-silico* oracle for the constraint. For the chemical formula, we simply parse the composition from the generated CIF. For space group determination, we use `pymatgen`’s `SpacegroupAnalyzer` with a precision of 0.2 angstroms [167]. For stability, we use M3GNet to estimate E_{hull} as before. Using the oracle’s labels, we then compute the percentage of cases in which the condition was properly met (Figure 5.6). The model is able to generate a material with the correct composition the majority of the time but becomes less reliable as the number of atoms in the chemical formula increases. Space group conditioning is more challenging, as it requires precise control and understanding of 3D structure, but the observed 24% is impressive when considering the 230 possible space groups. Generating stable/unstable structures as a binary task is the most challenging, likely because the training dataset is predominantly stable compounds and stability is defined only in reference to existing compounds. Stability is most easily controlled by modulating sampling hyperparameters.

INFILLING EXISTING MATERIALS In many practical settings, sampling and filtering materials from scratch is unnecessary. Good starting materials are often known, and manufacturing processes are easier to adapt to related compositions than develop completely from scratch by making small edits to their composition—often referred to as *template methods* [129, 182]. To emulate a typical template method, we construct a lookup table that maps each element to elements that have a similar atom radius when in the same oxidation state (code in Appendix C.3). We choose an element uniformly at random and swap it with a random element chosen from the table. The resulting structure is then relaxed using M3GNet. To improve this strategy using our fine-tuned LLM, we used the infilling prompt to obtain a distribution over elements (modulated with temperature τ) which we use instead of

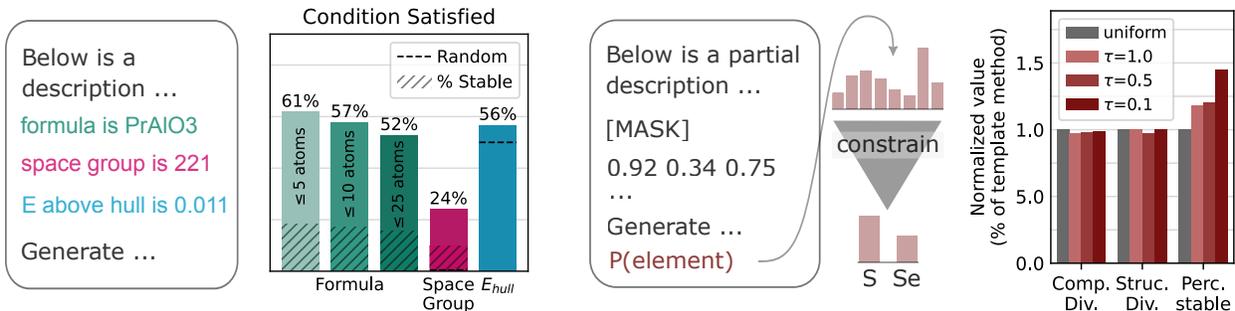


Figure 5.6: Text-conditional generation and infilling of existing structures with fine-tuned LLMs. **(left)** Including composition or property information (sampled from a hold-out set) in the text prompt leads to a high rate of samples with the desired composition/property (space group or stability). We bin stability as $\hat{E}_{\text{hull}} < 0.1$ (metastable) and $\hat{E}_{\text{hull}} > 0.1$ (unstable) for simplicity. Complex formulas and space groups challenge the model, but the samples are correct at a rate that facilitates practical use. We also show the rate of samples that both satisfy the condition and are predicted to be metastable by M3GNet. **(right)** Using the infilling prompt we can select mutations to existing materials. LLaMA-2 70B proposes a distribution over elements, which we constrain using knowledge of atom radii and charge interactions. We sample mutations with temperature τ and relax the results structure with M3GNet. When we apply this mutation procedure, we obtain more stable materials per mutation, with negligible changes to the overall diversity of the stable materials.

a uniform distribution over swaps. To evaluate our mutation procedure, we sample 3000 structures randomly from the test set and generate perform one mutation-relaxation step for each, using both uniform and language model-guided sampling. In Figure, 5.6 we show the percentage of stable compounds and diversity in the stable compounds for the uniform baseline and LLaMA-2 70B with different temperature values. LLaMA-2 70B proposes elements that lead to stable structures at a higher rate than the baseline template method without sacrificing diversity.

5.8 CONNECTIONS WITH SIMPLICITY BIAS AND LLMTIME

As crystal structures have translational symmetry, ideally our model’s likelihood should be invariant to translations. We propose *Increase in Perplexity under Transformation (IPT)* as metric for assessing the invariance of language models to continuous group transforma-

tions. For a transformation group G with group elements g and group action t , we define IPT for an input s ,

$$\text{IPT}(s) = \mathbb{E}_{g \in G} [\text{PPL}(t_g(s)) - \text{PPL}(t_{g^*}(s))]$$

where

$$g^* = \arg \min \text{PPL}(t_g(s))$$

and PPL is the perplexity of the sequence, the exponent of the length-normalized cross entropy loss, $\text{PPL}(s) = 2^{\text{CE}(s)/n}$. In our case G is the group of translation, where each g is a distance to translate by, and t_g is the mapping that decode the string, translates the coordinates (wrapping them around the boundary), and re-encodes the string. IPT captures the degree to which transformations change a language model’s compression ability. Good understanding of group transformations and invariance in the data should lead to minimal change in the perplexity of a transformed sequence. We can approximate IPT by sampling many values of g (e.g. 20), picking g^* as the minimum among those values, and computing a sample mean. Figure 5.7 shows the mean IPT of 500 random crystals from the test set, for each of the three LLaMA model sizes. We include additional details about our IPT calculation in Appendix C.2.5.

There are also natural connections between the hierarchical softmax distribution laid out in LLMTime [87] and our text encoding of crystals. In LLMTime, it was easy to define a continuous distribution over a bounded interval of the real number line by assigning a uniform distribution within the bucket corresponding a sequence of digits. Here we can likewise derive a continuous distribution for each atom location within the lattice (as defined by relative coordinates) by assigning a uniform distribution to each voxel defined by a sequence of

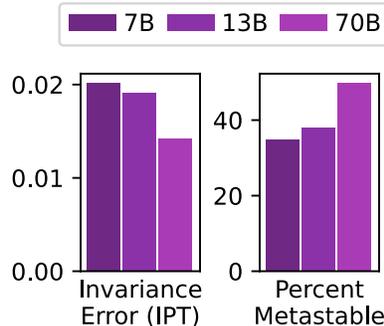


Figure 5.7: Translation invariance on test data and ability to generate stable materials increase in proportion. Larger models learn invariances from augmentations more effectively during training, likely as a result of their preference for abstract and compressible patterns.

(x,y,z) digits. Though we did not run the experiment, it could be interesting to compare the resulting likelihoods to ELBOs derived from competing diffusion models.

5.9 CONCLUSION

By generating a high rate of plausible stable materials (verified by DFT), we have demonstrated LLMs can be state-of-the-art generative models for atomistic domains with direct application of parameter-efficient instruction tuning and minimal task-specific modeling choices. This approach to generative modeling opens the door to multitask capabilities within a single sampling paradigm and multimodal training on atoms and text (e.g. to extract knowledge from a large corpus of scientific papers). We also advocate for the use of evaluation metrics (e.g. E_{null}) for generative models that are more closely tied to the downstream task of generating stable or metastable materials. The space of all hypothetical materials is combinatorially large (consider all the ways to pack 20 arbitrary elements into a box), but only a small subset of materials will actually be stable or metastable. Models that can directly generate near-stable structures make all downstream tasks far easier, and increases the likelihood the generative models may be useful for day-to-day tasks in materials discovery.

LIMITATIONS Our method shares the limitations of the underlying generative models. LLMs can be sensitive to precise details of the chosen prompt and the tokenization strategies, particularly in how tokenization affects processing of numbers. Hallucination of unphysical chemical elements or structures has been observed, though fortunately is easy to check and filter. Text-conditioning has the potential to tap latent conceptual understanding in the underlying LLM, but training LLMs that successfully leverage scientific and chemistry literature is a major outstanding challenge. Lastly, training the largest of our LLMs can be

prohibitively expensive for some computational budgets. Despite this, inference from all LLMs is often highly tractable when compared to baseline methods (Appendix C.2.7).

FUTURE DIRECTIONS There is substantial room for improvement in conditional generation, which could be used to directly generate materials with desired properties. While we did not pursue alternative sampling strategies in depth, approaches like classifier-free guidance [185] or variants of PPLM [57] might be useful in combination with fine-tuned LLMs to improve conditional generation. These methods could also be combined with primitives from Bayesian optimization for sample-efficient and uncertainty-aware design [89, 193].

6 | OPEN CHALLENGES IN APPLYING LANGUAGE MODELS TO NUMERICAL DATA

6.1 INTRODUCTION

Language modeling—training an autoregressive model on discrete tokens—is a surprisingly general paradigm, so far succeeding on text [30], images [238], videos [218], speech [40], time-series [9], and molecular structures [72, 86], as well as jointly on multiple modalities [42, 180]. For those working with scientific data, multimodality is particularly appealing because scientific systems (e.g. networks of interacting molecules) tend to have heterogeneous measurements and representations. A molecule, for example, is often associated with a set of atoms, with numerical positions and categorical identities (e.g., N or C), as well as assorted numerical and categorical properties, and potentially even extensive text descriptions. For this reason, significant interest has recently turned towards constructing models of molecules that can operate jointly over all data modalities [36, 55, 65, 100].

Despite these advances, it is easy to observe with simple experiments that language models still lag far behind other methods in modeling the properties of 3D structures. To fully realize the benefits of language models in these domains, we must therefore derive a

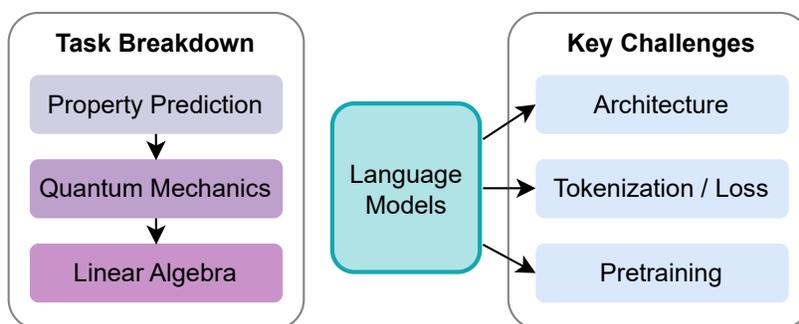


Figure 6.1: To study language models applied to property prediction, we break each into its constituent parts. We deconstruct property prediction by studying building block operations from linear algebra and understand language model performance through the lens of architecture, tokenization, and pretraining.

better understanding of the core challenges of this modality. While prior work has developed frameworks for thinking about language models applied to simple numerical tasks, like arithmetic and addition, little work has extended these theories to complex problems like property prediction.

To bridge this gap, we identify a few key building blocks of common property prediction methods, e.g. basic operations for linear algebra, and analyze how key modeling decisions affect when language models can learn these basic operations (Figure 6.1). From our experiments involving training thousands of language models to solve operations from linear algebra and simple physical modeling tasks, we draw the following conclusions:

- Contrary to work on arithmetic with transformers, we find that causal masking is *not* a fundamental bottleneck to current progress on 3D molecular data.
- When sweeping over many model choices, a model’s invariance to a problem’s symmetries is strongly correlated with predictive performance, showing that invariance is an important consideration.
- Discrete tokens incur a loss of predictive performance and exhibit poor scaling compared to alternative continuous losses.

- We observed limited transfer from text to property prediction. Large models pretrained on text often underperform small language models trained from scratch.

6.2 RELATED WORK

Learning simple arithmetic operations with language models is longstanding problem. Although addition and multiplication of integers or matrices are in the complexity class of algorithms that can be learned exactly by a transformer-based language model [157], they can still be difficult to learn in practice. Zhou et al. [255] speculate that some numerical operations are challenging to learn because they lack a *simple* program that can be expressed by a causal transformer. For example, addition of two multi-digit integers can be challenging when digits are ordered from most significant to least significant because causal attention has trouble building a representation for a carry operation. McLeish et al. [155] draw on similar observations to design an improved language model with expanded generalization abilities on addition and multiplication, by reversing number digits and providing special information about each digit’s location. While these studies provide useful frameworks for reasoning about the challenges of language models, they do not study high-dimensional objects, which have much more practical relevance. We show that in many cases the same intuitions do not naturally extend to our settings and different challenges dominate.

Going beyond integer inputs, Charton [35] shows that language models can learn basic operations from linear algebra like matrix addition, matrix multiplication, and eigenvalue computations. While we use some of Charton [35]’s tasks in our study and draw inspiration from their numerical string encodings, our analysis differs in fundamental ways because of our underlying motivation to approximate calculations from quantum chemistry. To this end, we focus much more on the invariance properties of learned models and introduce simple building blocks of physical models that are not studied by Charton [35]. In this way, our work

is more closely related to the work of Flam-Shepherd and Aspuru-Guzik [72], which shows that language models with standard training and simple tokenization methods can be used as strong generative models of 3D structures. However, like Alampara, Miret, and Jablonka [3], we also find that predictive tasks display different dynamics compared to generative modeling and that language models are not competitive with best-in-class predictive methods.

Because we apply language models to numerical data and closely study their interactions with choices in tokenization, we also draw on the work of Golkar et al. [83], which introduces a continuous alternative to discrete tokenization in language models (xVal) on mixed categorical and numerical data. xVal models all numbers with a single token and uses a single weight and bias vector for inputs and outputs, instead of an embedding matrix for many numerical tokens. xVal is thus akin to transformers applied directly on continuous inputs or to graph neural network methods applied to numerical prediction problems. We find that xVal does lead to improvements in many settings and therefore provides valuable perspective on what facets of language modeling are most challenging when learning on geometric data.

As we are also interested in how text pretraining can act as a useful inductive bias, our work also intersects with work that applies text-pretrained models to zero-shot prediction on other modalities [86, 104]. Although text pretraining holds the potential to help models learn general-purpose circuits over discrete sequences, we find that it is ultimately unhelpful in our considered numerical tasks.

6.3 PRELIMINARIES

6.3.1 MOTIVATING PROBLEM SETTING

In Table 6.1, we show how a basic language model architecture [208] trained from scratch compares to popular and state-of-the-art methods on a popular molecular property bench-

mark, predicting the highest occupied molecular orbital (HOMO) of organic compounds in the QM9 dataset [179]. It is easy to see that language models are an order of magnitude worse than competing models, and in the following sections, we will try to articulate a few possible reasons for this large performance gap.

Table 6.1: Comparison of popular and state-of-the-art approaches for predicting HOMO on QM9.

Method	MAE (↓)
LLaMA (from scratch)	212 meV
Non-equivariant GNN	71.4 meV
Equivariant GNN [187]	51.9 meV
JMP-L [191]	8.8 meV

6.3.2 COMPUTATIONAL CHEMISTRY AS A TEST BED

To develop a better mechanistic understanding of the challenges of applying language models to tasks like property prediction, let’s break down this goal problem into its constituent parts and see where language models run into trouble. One core task in quantum chemistry is calculating the energy of a configuration of many atoms. Low energy configurations are stable and practically useful, for example in novel materials or the binding interface of therapeutic drugs. Atomic nuclei can be modeled as point charges,

$$E_{\text{nuc}} = \sum_{i < j} q_i q_j / D_{ij} \tag{6.1}$$

where D is the distance matrix between nuclei and q is the charge of each nucleus. To model electrons, however, more complex methods are needed, for example, Hartree-Fock, which iteratively solves¹

$$F(C) C = C\epsilon \tag{6.2}$$

¹We show the Roothaan equations using an orthonormalised basis set

where F is the Fock matrix, C are the orbital coefficients and ϵ is a diagonal matrix of molecular energies. At each step, C and ϵ are obtained by solving a generalized eigenvalue problem using F constructed from the last approximation of C , and, upon convergence, the electron energy is

$$E_{\text{elec}} = \text{Tr}(\epsilon) + \text{Tr}(C^\dagger H C)$$

where H is the system's Hamiltonian (constructed using the position and charge of the atomic nuclei). From this basic calculation, it's clear there are at least a two key components:

- Calculating and working with **distances** on coordinates.
- Linear algebraic operations, such as **matrix addition** and **multiplication**, as well as more complicated procedures like **solving for eigenvalues**.

Ultimately these are the building blocks that we will study in depth.

Of course, one of the strengths of neural networks is their ability to approximate expensive procedures with a fixed computation budget (one forward pass during training), so we do not necessarily expect language models to recapitulate tools from mathematics and physics. We can, however, use existing methods for approximating physical observations as a way of debugging current limitations of language models, if not to perfectly understand their internal mechanisms. If language models struggle to learn matrix multiplication but not matrix addition, for example, we can speculate that the scalar multiplication of many operands might be a roadblock, and we can work on this limitation directly.

6.4 THEORETICAL LIMITATIONS

In a single forward pass of a neural network, there are fundamental limits on both (1) the number of serial operations and (2) the amount of memory for intermediate results. When the number of steps that must be performed serially exceeds the depth of the network, the

network will not be able to learn the exact function. Therefore, one forward pass of a network with fixed depth will inevitably fail on problems that require a number of sequential steps that grows with the input. Notably, this limitation is not relevant for the many functions that can be parallelized, like matrix multiplication, which is in TC^0 and therefore should be possible to learn with a single forward pass [157].

Like serial computation, memory can also be a bottleneck, as functions involving the matrix product $A^T A$ where A is $m \times n$ with $n \gg m$. Storage of the intermediate result requires $O(n^2)$ memory, but the input size only scales like $O(n)$. When n is greater than the depth of the network, there can be challenges in storing all intermediate computations in the network’s activations.

6.5 PRACTICAL CHALLENGES

In practice, reasoning about what solutions tend to be learned by a particular architecture is often more important than fundamental constraints. Even when a function can be represented in the function class, the statistical nature of the problem and questions of approaching the solution via approximations can play a larger role. Aside from not being able to represent the exact algorithm, why would autoregressive language models be limited in solving these numerical problems? We formulate 3 hypotheses and in the following sections provide evidence for and against each of these hypotheses.

1. *Model architecture* (section 6.7): Features in autoregressive models are unidirectional (causal masking), which makes learning some numerical operations challenging. For example, when digits are passed from left to right into a language model, it is challenging to express the addition of two numbers using a carry bit. Structured numerical data also often obey constraints that are easy to express analytically (e.g. invariance to rotations). Incorporating these constraints can make learning more sample-efficient or

improve generalization [75], but language models are typically unconstrained. In both cases, architecture decisions might have a significant affect on test performance.

2. *Tokenization of numbers* (section 6.8): Tokenization can lead to strange artifacts in text-pretrained language models [30, 215] and hinder their application to numerical inputs [88]. While language models trained on numerical data often explore multiple tokenization schemes [35, 72, 83], they are often presented as ablations rather than analyzed in their own right.
3. *Too little data or pretraining* (section 6.9): Data for some numerical tasks can be relatively limited or extremely noisy, making language models less likely to succeed compared to models with more domain-specific assumptions. Other works show that text pretraining can serve as a surrogate for domain-specific pretraining or inductive biases [86].

6.6 EXPERIMENTAL SETUP

To test our hypotheses, we train thousands of language models that vary in model architecture, model size, tokenization method, loss function, and pretraining method.

STRING-ENCODING AND TOKENIZATION To turn numbers into tokens, we convert all numbers to a fixed precision and then convert these numbers to variable length strings by ignoring any leading zeros. These strings are then tokenized using a vocabulary of all numbers up to a certain chunk length, for example, {"1", "2", ..., "998", "999"} for a chunk length of 3. We greedily select the largest subsequence from right to left. For negative numbers, each negative number is prepended with "-". These strategies are similar to P10, P1000, and FP15 in Charton [35], but, in our case, we choose to drop the exponent term used by Charton in favor of variable length because our inputs do not contain many different orders of magnitude. In addition to standard tokenization with an embedding matrix, we also explore

Abacus embeddings [155] and xVal [83], which are tokenization methods specifically designed for processing numbers.

MODELS We present results for both language models trained from scratch and frontier language models pretrained on text. Pretrained models can reuse general computational circuits and features developed on the pretraining text data, but may not be as well suited for numerical data in the given format. When training models from scratch, we use the LLaMA-2 [206] architecture with between 4 and 8 layers and hidden size 512, which translates to between 20 million and 50 million parameters. We train models with a learning rate of 0.0001 or 0.0005 and a cosine schedule. When studying pretrained models, we use LLaMA3.1-8B [66], and the default LLaMA-3 tokenization, which, on numerical inputs, is identical to our 3-digit chunking method. We fine-tune the LLaMA3.1 models using LoRA with rank 8 and alpha 32 for one epoch. To make predictions with the models, we draw 10 samples at temperature 1 and calculate the median at each dimension of the output.

DATASETS Our datasets are chosen to represent building blocks of common functions on numerical data. They have varying degrees of difficulty, with some being computable exactly by transformers while others can only be approximated. We explore two categories of tasks:

- **Linear algebra:** Following [35], we create $n \times n$ matrices with $n \in [2, 10]$ and evaluate (a) matrix addition (b) matrix multiplication, and (c) calculating real eigenvalues. We train on matrices of mixed sizes, with a distribution of n weighted n , so that we train on more large matrices. The input matrices have coefficients sampled uniformly from $[-10, 10]$, and resulting eigenvalues having a center distribution with standard deviation $\sigma = 10\sqrt{n/3}$.

These tasks have significant variations in difficulty. While matrix sum and product are computable in theory by a language model, computing an eigenspectrum is not and is more

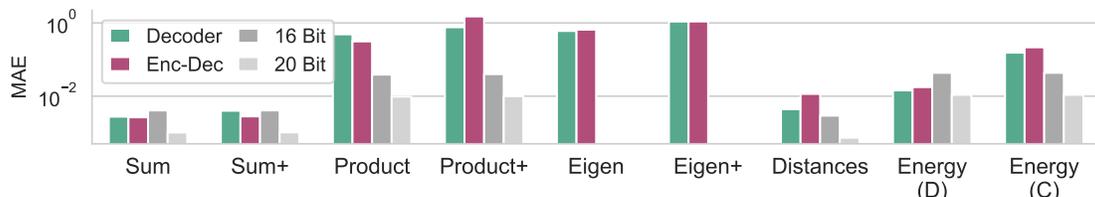


Figure 6.2: Encoder-decoder architectures have theoretical advantages over decoder-only architectures on our numerical tasks, but we find in practice the difference is minor. In our encoder-decoder models, layers are split equally between the encoder and decoder. A task name with ‘+’ indicates a holdout of unseen matrix shape—a harder test of generalization. We include quantized numerical operations as baselines. 16 bit refers to a quantized operation with an 8-bit mantissa and 8-bit exponent. 20 bit has a 10 bit mantissa and 10 bit exponent. We do not provide results for a quantized eigenvalue solver because PyTorch does not provide an easy mechanism for constructing one.

intrinsically serial than sum and product, making it more challenging for transformers. In addition to testing on matrices drawn from the same distribution as the train data, we also include a special generalization setting (marked with ‘+’) in which we train on $n \in [2, 10] \setminus \{8\}$ and evaluate on $n = 8$. While past research often tests generalization by evaluating on problems strictly larger than the problems in the training datasets [255], we opt for an interpolative setting because it is less confounded by the inherent limitations of position embeddings and reflects other facets of generalization on numerical data.

- **3D structures:** Using the data from QM9 [179], we evaluate on the highest occupied molecular orbital (HOMO) regression task. We also evaluate on a set of simpler functions on QM9 coordinates, including (a) calculating a distance matrix on coordinates and (b) calculating a simple potential energy over the atomic nuclei. For the potential energy task, we test on either pre-computed distances or directly on coordinates, which disentangles the challenge of computing distances internally within the neural network, a task that can involve storing an intractable number of intermediate variables. Alongside the linear algebra tasks, these problems encompass many of the fundamental operations of quantum chemistry. It might be difficult to approximate current computation methods without being able to express reasonable approximations to these simpler problems.

For linear algebra tasks, we use 500,000 training examples, and for 3D structures, we use

100,000 examples. We use 400 fixed test points for all evaluations.

BASELINE METHODS Our first baseline is low-precision quantization of the floating point numbers used in the correct computation within the synthetically generated tasks. We know that transformers struggle with performing exact arithmetic, even for integers, therefore we should expect that arithmetic will at best be performed approximately within the transformer. This quantization baseline evaluates the impact of using a correct algorithm but with only limited precision. We use QPyTorch [246] and allocate an equal number of bits to the exponent and mantissa. Our two quantization baselines use 16 and 20 total bits, and this sets a reasonable ceiling on model performance.

Our other baselines are equivariant graph neural networks (EGNNs) [187], which learn functions that are equivariant to permutations, rotations, and translations. EGNNs are therefore particularly useful in understanding how symmetries affect performance on our tasks. Following the original EGNN experiments on QM9, we use networks with 7 layers and hidden dimensions of size 128. Training details are included in Appendix D.3.

6.7 MODEL ARCHITECTURE

CAUSAL MASKING In addition to whether or not the input x is featured in the loss or is masked out, the decoder-only autoregressive structure of the language model has an impact on which operations are easy to express. For example, as identified by Zhou et al. [255], the carry used in adding two numbers is a useful intermediary for the task, but if the numbers are ordered most significant digit first then its computation is nontrivial. In the structured numerical data context, an analogous challenge might arise when outputting scalar values which depend on aggregating information from input set data. For example, with an input $X = \{x_1, x_2, \dots, x_n\}$, computing $y = \sum_{i,j} K(x_i, x_j)$. As a quadratic time

operation that depends on all pairs of inputs, it might seem difficult for a causally constrained model to perform this computation within the linear space allotted to the model. When additional space is provided, such as by having the model output all pairs of $K(x_i, x_j)$ in token form as an intermediate result, then the computation can be performed easily within the layers of a causally masked transformer. We demonstrate that this is the case in [section D.8](#), providing a RASP-L program [255] which computes all pairs distance (RASP-L is a restricted programming model designed to match the readily learnable operations within a transformer). Reasoning models could plausibly learn to break up the problem in this way to circumvent these challenges, but it is unlikely that they do so presently.

To test the hypothesis about the added difficulty from the causal masking, in [Figure 6.2](#) we compare the performance of a decoder-only model with loss masking to an encoder-decoder approach where only y is modeled autoregressively and X can be attended to bidirectionally by the encoder. We find that, contrary to intuitions, encoder-decoder models do not perform significantly better than models with only causal masking. To enable an apples-to-apples comparison in these experiments, we use the same number of parameters in each architecture for each of three fixed parameter counts. In causal models, every layer is causal, whereas in encoder-decoder models, half the layers are in a bidirectional encoder and half the layers in a causal decoder. For tasks with a complex and high-dimensional output, it is possible that having a limited number of decoder layers could have a negative impact on the coherence of the output relative to a decoder-only architecture. This is one possible explanation for encoder-decoder architecture’s significantly worse performance on calculating distances, where the output is a flattened upper triangular.

As a small additional experiment, we also explore McLeish et al. [155]’s approach to tokenizing numbers, which involves reversing the digits in number allowing for simpler algorithms implementing arithmetic operations Zhou et al. [255]. In addition to reversing the digit, a special embedding is added to identify each digit position within a number. Un-

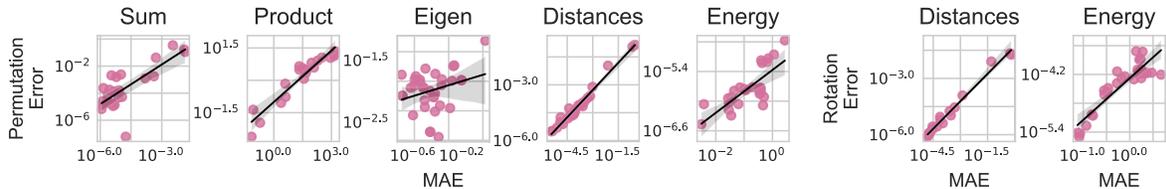


Figure 6.3: (Top) Degree of invariance (permutation or rotation error) strongly correlates with the ability to fit the task (MAE) across several model sizes, tokenization methods, and training runs. Results are displayed with both axes log-scaled. For rotation invariance, we only study tasks on 3D structures. Shading is a 95% confidence interval for the regression.

like the original paper, however, we use a plain decoder-only transformer model without parameter-sharing or skip connection to the input. In Table 6.2, we show that this intervention has a negligible or slightly negative effect overall. Although McLeish et al. [155] designed their approach with large multi-digit numbers in mind, it is surprising that there is no positive effect on learning operations that depend on addition and multiplication as a subroutine. Together, these two results (comparing architectures and input orderings) suggest that artifacts of causal masking are likely not the largest bottleneck to language model success on our tasks.

Table 6.2: Digit order has a negligible effect on relative error. We report geometric mean across tasks with standard errors.

Method	MAE (\downarrow)
Base	0.237 ± 0.12
Reversed	0.309 ± 0.14

SYMMETRIES Symmetries can be hard-coded into a model’s architecture, but this process is not common in language modeling applications and is challenging when operating on tokenized strings. In this section, we explore how language models learn invariances or equivariances on our numerical tasks and quantify how correlated learning symmetries is with predictive performance. In our linear algebra tasks, the most relevant symmetry is over the symmetric group, i.e., permutations of the rows or columns. For any permutation

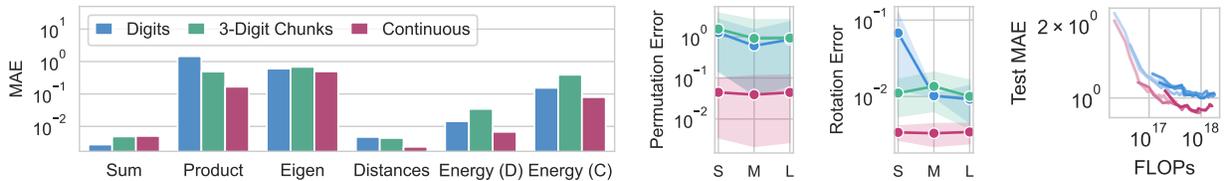


Figure 6.4: (Left) We train causal transformers with different tokenization schemes and witness a significant advantage from learning a continuous prediction head. By contrast, differences between discrete tokenization schemes (digits vs. chunks) are inconsistent with multi-digit schemes performing better on some tasks and worse on others. (Center) Using a continuous prediction head leads to higher invariance at smaller model sizes. For discrete methods, larger models are required to learn invariance. Numbers are the geometric mean over tasks, and shading denotes a 95% confidence interval. (Right) We present a scaling experiment on the matrix product task. We train 5 different model sizes, for both standard digit tokenization and a continuous prediction head. Shading denotes parameter count, while FLOPs are a function of parameter count and gradient steps. Additional details are provided in [section D.5](#).

matrix P and square matrices A, B , equivariant matrix operations like addition or matrix multiplication produce a permuted result: $(PAP^{-1})(PBP^{-1}) = P(AB)P^{-1}$, whereas as other operations like computing the eigenvalues are invariant to permutations: $\lambda_i(PAP^{-1}) = \lambda_i(A)$. These symmetry transformations are merely linear transformations of the input matrix, and can be explicitly written as such: $\text{vec}(PAP^{-1}) = P \otimes P \text{vec}(A)$, where vec flattens the elements of the matrix into a vector and \otimes is the Kronecker product.

Our 3D structure tasks are equivariant or invariant to permutations (over the atom indices) as well as rotations, and translations on the coordinates of the atoms. We quantify invariance by calculating the predictions of the model for 10 examples transformed with random permutations or rotations. The invariance error is measured as the standard deviation per dimension normalized by the absolute value of the ground truth values and averaged over all dimensions: $\text{Err} = \frac{1}{D} \sum_{i=1}^D \frac{\sqrt{\text{Var}_{M \sim \mu}[f_i(Mx)]}}{|f_i^*(x)|}$. Here, μ represents the uniform distribution over the group (formally the Haar measure), M is a matrix (of the relevant representation) for the group, and f^* is the function we are trying to fit. Following standard practice, we train our 3D structure models with rotation augmentations, and we also add permutation augmentations. For linear algebra tasks, we do not apply augmentations.

In [Figure 6.3](#) (top), we show the correlation between predictive performance (relative error) and invariance to permutation or rotations. The points displayed are models that vary in size, architecture, and training hyperparameters. Across almost all tasks, there is a strong correlation, indicating that good models also tend to be invariant models. The notable exception is solving for eigenvalues, which displays the opposite trend, likely due to a spurious correlation between the matrix ordering and eigenvalue spectra in the training dataset. In many cases, the best models can approach perfect invariance, with invariance errors on the order of 10^{-6} . Yet, even when models are nearly invariant, small changes in invariance appear to be correlated with improvements in performance.

To further explore the impact of equivariance, we compare decoder-only language models trained on digit tokens against GNNs with and without rotation equivariance in [Table 6.3](#). GNN indicates permutation equivariance, while EGNN indicates permutation, translation, and rotation equivariance. The evaluation tasks are energy (from coordinates) and HOMO, both of which have permutation and rotation symmetry. In the results, invariance/equivariance again has a strong connection with predictive performance. Combined with the results above, we can conclude that invariance has a clear connection with performance on our numerical tasks. Surprisingly, language models can achieve high levels of invariance, but these high levels do not appear to saturate predictive performance.

Table 6.3: GNNs outperform LMs on the energy prediction task (from coordinates) and benefit from equivariance.

Method	MAE (\downarrow)
LM	0.209
GNN	0.079
EGNN	0.041

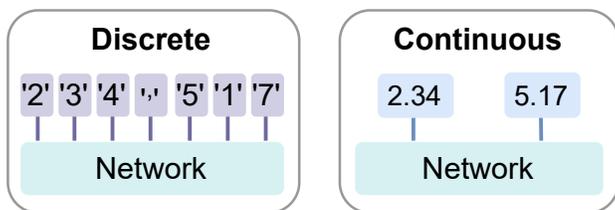


Figure 6.5: We can model numerical data as strings or as vectors of continuous values. Each approach has unique costs and benefits.

6.8 TOKENIZATION

Besides architecture and training loss, tokenization is the other key design decision in constructing language models (Figure 6.5). When training on text, most language models employ tokenizers that compress commonly occurring sequences of bytes (e.g. byte-pair encoding). However, naively applying these same tokenization methods to numbers can lead to problems, because small changes to the value of the number can lead to large and hard to model changes in the tokenization of the number string [88]. Character-level or n-gram tokens, therefore, are popular choices when modeling numbers, but while many papers employ these methods [72, 249], there is little understanding of how tokenization affects the model’s ability to learn basic numerical operations.

We might also ask why it is even desirable to work with discrete tokens in the first place, as molecular structures and properties are typically represented with continuous vectors. To this end, we consider a few a-priori trade-offs and show that there are reasons for supporting either approach.

CONTINUOUS SEQUENCES: Each number is a floating point value, typically at the same precision as the weights of the neural network.

- **Pros:** (a) domain-specific properties (e.g. invariance/equivariance) have simple relationships with the model parameters. (b) order information is preserved in the input and in

loss functions. (c) it is not necessary to learn an embedding matrix, or associated linear layers, which might be very large.

- **Cons:** (a) information contained in the scale of the numbers can be destroyed by normalization used to improve numerical stability. (b) modeling numbers of radically different scales can lead to numerical instability in the input. Transforming with log and exp can stabilize the input but have poor gradient behavior. (c) multimodality (mixed categorical and continuous variables) in the output space can be hard to represent.

DISCRETE SEQUENCES: Each number is converted to a string and then to a sequence of tokens, i.e. [“1”, “.”, “5”, “6”], and corresponding integers.

- **Pros:** (a) distributions on sequences are densities over numbers without strong distributional assumptions or complicated losses. (b) input numbers do not need to be normalized. Numbers can in principle be large or small without causing fundamental problems, though length generalization is not guaranteed.
- **Cons:** (a) learning basic operations on numbers might require many samples because of a large vocabulary and complicated algorithms for operating on strings. (b) hallucination of non-number outputs.

To test the empirical effects of tokenization, we compare a few approaches, including both discrete and continuous losses. In the discrete setting, we try both tokenizing individual digits and tokenizing in 3-digit chunks. When using chunks, we always tokenize from right to left to maintain a consistent token meaning for strings of different lengths. In principle, the primary trade-off between these approaches is between vocabulary size and sequence length, as chunked sequences are shorter but might require a larger training dataset to cover the space of 10^k tokens, for chunk size k , some of which might occur rarely. In addition to these two discrete approaches to processes numbers, we also run experiments with xVal [83], which replaces discrete vocabularies and their associated embedding with a single linear

projection that turns scalar inputs to vectors the same dimension as token embeddings and which projects final output layers. Instead of the cross-entropy loss, xVal uses an L2 loss on its continuous prediction. xVal is a useful counterpoint to purely discrete approaches because it sidesteps several key challenges of tokens, for example learning large vocabularies, long sequences, and potential challenges in learning symmetries.

In [Figure 6.4](#) (left & center), we show the effect of tokenization on predictive performance and symmetry learning. Overall we see that adopting a continuous approach leads to lower errors and more invariant predictors. By contrast, the difference between character-level (digit) and n-gram (3-digit chunk) schemes is inconsistent in terms of errors and nearly equivalent in terms of invariance. The latter result is surprising given our relatively large datasets, which provide reasonable coverage of the tokens in the vocabulary. The relationship between invariance and model size in xVal hints that maybe using any discrete representation incurs significant overhead to learn the appropriate structure, as xVal appears to acquire much higher levels of invariance for all model sizes. In [Figure 6.4](#) (right) we also show that xVal exhibits an improved scaling law on the matrix product tasks (details in [section D.5](#)).

6.9 TEXT PRETRAINING

As we’ve seen so far, language models typically require large model sizes in order to capture invariances and make good predictions. For sufficiently large datasets, this allows language models to perform on par with hand-crafted methods, but in other cases these extra parameters lead to poor generalization or slower convergence for fixed compute. The typical solution for this problem is extensive unsupervised pretraining, which can unlock the benefits of language modeling, while matching performance on narrow tasks. Prior work shows that text pretraining can serve this role in some cases. For example, Delétang et al. [\[58\]](#) and Goldblum et al. [\[82\]](#) show that text-pretrained models are general-purpose

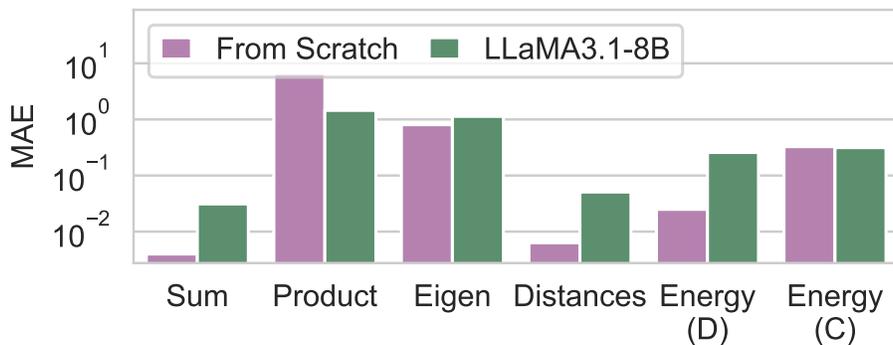


Figure 6.6: We compare our small language models trained from scratch with finetuned versions of large text-pretrained models. Text-pretrained models perform worse on every task except matrix products, which might benefit from the pretrained model’s additional capacity and ability to model high-dimensional outputs.

compression engines that can match domain-specific compression on non-text modalities. In practical applications, results Gruver et al. [86] find that text-pretrained models are useful initialization for generating 3D crystal structures, but in some cases, authors have also found negative transfer from text-pretraining [3].

To explore pretrained models, we compare our small from-scratch models with LLaMA3.1-8B, which we fine-tune for one epoch. Although there are fewer gradient steps in the case of finetuning, the compute requirements are roughly equal because of the much larger parameter count in the pretrained model. As with models trained from scratch, we make predictions by drawing 10 samples and taking the median per dimension. Figure 6.6 (right) shows the resulting errors, for which pretrained models have worse performance in all tasks but matrix multiplication. We posit two possible explanations for this discrepancy: (a) matrix product requires the most capacity to learn effectively (b) matrix product has very high-dimensional outputs consisting of matrices containing large numbers, and text-pretraining is primarily helpful in modeling patterns in long sequences. If this were true, however, we might also expect some benefit on matrix addition and computing distance matrices.

6.10 CONCLUSION

In this chapter, we explored several explanations for the subpar performance of language models on 3D property prediction tasks. Through interventions like modifications of the architecture and loss function, we see that some of the explanations are not supported by the data, while others, such as the importance of invariances, hold up to scrutiny. Our experiments suggest that language models converge to increasingly accurate and nearly invariant solutions when given sufficient model capacity and yet still have a large gap when compared to a method like xVal. One exciting avenue for future work is to expand upon the xVal approach to support multimodal generation, for example with score matching.

DISCUSSION: LANGUAGE MODELS FOR NUMERICAL DATA

In Chapters 4, 5, and 6, we explored successes and failures in applying language models to numerical data. We showed that text pretraining often induces a bias for simple functions that can encourage sample-efficient learning and good generalization. We also showed that autoregressive models on sequences of digits can be expressive distributions over both 1D and 3D data, and, by extension, over high-dimensional data expressed as sequences of floating point numbers or 3D coordinates. We showed in two practical cases that these strengths of language models can lead them to perform competitively against domain-specific models in zero-shot of finetuning settings. Finally, we also showed that text-pretrained language models are not a panacea and can lag behind domain-specific models dramatically in some cases, such as molecular property prediction. We speculated on the origin of this performance gap and identified the use of discrete tokens as one possible explanation, due to its empirical relationship with symmetry learning and the empirical relationship between symmetry learning and generalization. This work, however, is ongoing, and more research into the failure modes

of language models and their origins would significantly benefit the community.

Language models have the potential to be general-purpose foundations for not just text but also numerical data, enabling data-efficient learning, generalization across data sources, and enabling new capabilities, such as generation conditioning on text or reasoning chains. Some fraction of machine learning research in the near term will thus concern how close we are to reaching this potential, or whether the role of language models will consist instead in solely imitating the text-based work-flows of humans, such as in agents that write or train their own models. Our work suggests that there is some promise in using LLMs as foundations for other modalities, but that it is far from being the definitive solution.

7 | CONCLUSION

This thesis has demonstrated how pretraining can generate useful inductive biases that reduce our reliance on domain-specific architectural constraints, enabling general frameworks like autoregressive sequence modeling to tackle diverse numerical problems. At the same time, our analysis of Hamiltonian neural networks and convolutional neural networks reveals that while inductive biases are often misunderstood, they remain crucial for model scaling and generalization. The path forward likely requires balancing simplified architectures with carefully chosen constraints - we cannot yet reduce all modeling tasks to fine-tuning a single pretrained model. These results provide insight into the transition from specialized deep learning approaches toward large-scale pretraining, highlighting both the opportunities and limitations of more general architectures.

FUTURE WORK

AGENTIC SYSTEMS AND MULTIMODAL FOUNDATION MODELS

The landscape of artificial intelligence is evolving beyond narrowly defined tasks toward systems that can emulate complex human behaviors. This evolution raises interesting questions about the relationship between model designers and the models themselves, as these systems increasingly demonstrate the ability to utilize computational resources and execute sophisticated sub-tasks autonomously.

Our work on language models in time series forecasting initially focused on their ability to incorporate textual information into predictions. However, this capability hints at broader potential: future language models might autonomously construct and deploy traditional statistical tools like seasonal ARIMA models, design custom Gaussian process kernels, or develop neural network architectures based on contextual understanding. These systems might even mirror a data scientist’s workflow, fitting multiple models, analyzing their predictions, and forming intuitive judgments based on the results.

This trajectory raises fundamental questions about the future of machine learning system design. We must carefully consider which problems are best solved through traditional differentiable function approximation from input to output, and which require more complex approaches using reinforcement learning to develop sequences of actions. While related, these approaches offer different trade-offs and capabilities. Similarly, we must examine whether architectural properties like invariance and equivariance will remain optimal for certain problems, particularly those where exact symmetries are fundamental to valid solutions. Even in a future dominated by general AI systems, these specialized architectural designs might persist as crucial subsystems for specific tasks.

AUTOREGRESSION OR DENOISING FOR FOUNDATION MODELS

In the work covered in this thesis, we primarily covered the use of autoregressive language models as general-purpose foundations, but there are alternative paradigms with different trade-offs. For example, diffusion, flow-matching, and consistency models have thus far been more successful for image data and can generate high-quality samples with only 10-100 forward passes of the model. Non-autoregressive models are also not constrained to one ordering and therefore might not incur the downsides of causal masking discussed above. Diffusion language models are increasingly popular and are closing the gap that once existed between their performance and traditional autoregressive models [12, 183]. Diffusion language models

promise to be significantly faster than autoregressive models and might enable multimodal modeling with diffusion losses on each modality, as has emerged in generative modeling of protein sequence and structure [32, 99]. While non-autoregressive models offer fast sampling and historically better sample quality on non-text modalities, they might also be fundamentally weaker in terms of sample-efficiency than language models because of strengths of causal masking combined with teaching forcing, which allows autoregressive models to learn on all noise levels for each example at each training step. It's possible that this strength combined with the future image/video tokenizers will be sufficient to make autoregressive models the dominant paradigm, but only time will tell.

EXPLORATION WITH LANGUAGE MODELS

In the scientific community, there is increasing excitement about using language models to propose and test hypotheses, including promising initial results [115, 144]. Most attempts thus far do not account for risk and resource allocation, which are critical aspects of scientific research that slow progress. In other instances of sequential decision making under uncertainty, these challenges are handled by strategically balancing exploration (to increase understanding of the environment) and exploitation (to make progress with existing knowledge). It's unclear if language models learn to recapitulate human strategy for exploration, and there is some evidence that they do not [162]. Much of the work in my PhD was focused on applications of Bayesian optimization to protein design, though this work did not end up in this thesis, and in Bayesian optimization exploration is encourage through modeling epistemic uncertainty. There is some indication that language models can model their own uncertainty, though it does not emerge for free from pre-training [124]. It would be very interesting to gauge whether these uncertainties can be used for efficient exploration or decision making, particularly in scientific applications.

A | APPENDIX: DECONSTRUCTING LEARNED SYMMETRIES

A.1 APPENDIX OUTLINE

This appendix is organized as follows. In Section B, we present proofs for the energy conservation properties of HNNs and Neural ODEs, as well as a more detailed description of symplecticity. In Section C, we described the details of our experiments on Mujoco, including our data preprocessing, training, and architectural decisions. Lastly, in Section D, we provide additional experimental results requested by reviewers of our original submission. These include a comparison of alternative loss functions and a comparison on additional rigid body systems.

A.2 MATHEMATICAL DETAILS

A.2.1 ENERGY CONSERVATION FOR NEURAL ODES

Let $F = J\nabla H$ be the ground truth dynamics of a time independent Hamiltonian, and \hat{F} be the dynamics learned by a neural network through an learned Hamiltonian \hat{H} for HNNs or otherwise. Given some initial condition z_0 , let \hat{z}_T denote the solution to $\dot{z} = \hat{F}(z)$ at time

T starting from z_0 and z_T be the solution to the ground truth dynamics from z_0 .

Suppose the error in the dynamics model $e(z) = \hat{F}(z) - F(z)$ is bounded $\forall z : \|e(z)\| < \delta$ and that we are only considering a bounded region of the state space (such as the states of a pendulum with bounded energy).

Since energy is conserved $H(z_t) = H(z_0) = H(\hat{z}_0)$, we can write the the energy error

$$H(\hat{z}_T) - H(z_T) = H(\hat{z}_T) - H(\hat{z}_0) = \int_{\hat{z}_0}^{\hat{z}_T} \nabla H(z)^\top dz$$

Since the value is independent of the path, we may consider the path given by the approximated dynamics \hat{z}_t . Noting that $d\hat{z}/dt = \hat{F}(\hat{z}_t) = J\nabla H(\hat{z}_t) + e(\hat{z}_t)$, we have

$$\begin{aligned} H(\hat{z}_T) - H(z_T) &= \int_0^T \nabla H(z)^\top \frac{d\hat{z}}{dt} dt = \int_0^T \nabla H(\hat{z}_t)^\top \hat{F}(\hat{z}_t) dt \\ &= \int_0^T [\nabla H(\hat{z}_t)^\top J\nabla H(\hat{z}_t) + \nabla H(\hat{z}_t)^\top e(\hat{z}_t)] dt \\ &= \int_0^T \nabla H(\hat{z}_t)^\top e(\hat{z}_t) dt. \end{aligned}$$

Bounding the maximum value of the integrand along the path, we have that

$$|H(\hat{z}_T) - H(z_T)| < T\delta \sup \|\nabla H\|, \quad (\text{A.1})$$

which grows only linearly with time.

This linear bound on the energy error is in stark contrast with the state error which could grow exponentially according to the Lyapunov exponents, even if the dynamics error is arbitrarily small. Advancing the ground truth and learned dynamics forward to some small time $t = \epsilon$, $\hat{z}_\epsilon = z_0 + \epsilon\hat{F}(z_0)$ yields error $\|\hat{z}_\epsilon - z_\epsilon\| = \|\epsilon\hat{F}(z_0) - \epsilon F(z_0)\| = \epsilon\|e(z)\| < \epsilon\delta$. And yet, this error even if propagated by the ground truth dynamics will grow exponentially $\|\hat{z}_T - z_T\| \approx e^{\lambda T}\|\hat{z}_\epsilon - z_\epsilon\| = e^{\lambda T}\epsilon\|e(z)\|$

A.2.2 HNN ENERGY CONSERVATION

A simple but erroneous argument for why HNNs approximately conserve the true energy goes as follows:

We would like to know if HNNs achieve better energy conservation given the same levels of error in the predicted dynamics. For HNNs, $\hat{F} = J\nabla\hat{H}$, and we can see that the dynamics error $e(z)$ can also be written as $e(z) = J\nabla(\hat{H} - H)$.

If we could convert a bound on the derivatives $\|e\| = \|\nabla(\hat{H} - H)\| < \delta$ (since J is an orthogonal matrix) into a bound on the learned Hamiltonian itself $E(z) := \hat{H}(z) - H(z) - c$ and $|E| < \Delta$ holding globally for some constant c , then we would have a constraint on the energy error that doesn't grow with time. Expanding the difference in initial and final energy, the constant c cancels out and we have

$$\begin{aligned} H(\hat{z}_T) - H(\hat{z}_0) &= \hat{H}(\hat{z}_T) - \hat{H}(\hat{z}_0) - E(\hat{z}_T) + E(\hat{z}_0) \\ &= -E(\hat{z}_T) + E(\hat{z}_0), \end{aligned}$$

using the fact that the learned energy function \hat{H} is conserved over the model rollout \hat{z}_t . If there was a constraint $|E| < \Delta$ then

$$|H(\hat{z}_T) - H(\hat{z}_0)| < 2\Delta.$$

Unfortunately, even if the gradients are close and δ is small, that does not imply that Δ is small. Small differences in gradient can add up to very large differences in the values of the two functions. While the dynamics may well approximate the data, and achieve low rollout error, there is no reason to believe that at a given point in phase space the learned Hamiltonian should have a value that is close to the true Hamiltonian.

A.2.3 SYMPLECTICITY

Symplecticity is the requirement that the dynamics satisfy $(JDF)^\top = JDF$. Defining $G = JF$, the requirement is simply that the antisymmetric part of the jacobian is 0, $DG^\top - DG = 0$.

Unpacking Poincare’s lemma requires some familiarity with differential geometry concepts such as differential forms and exterior derivatives, and so we will assume them but for this section only. Poincare’s lemma states that on a contractible domain (such as \mathbb{R}^n) if a differential k -form ω is closed $d\omega = 0$ (the exterior derivative of ω is 0) then it is also exact $\omega = d\nu$ (it is the exterior derivative of another differential $(k-1)$ -form ν). While F is a vector field, $G = JF$ is a differential 1-form (dual to a vector field). If DG is symmetric, then it is also closed: $dG = \sum_i \partial_i G_j dx^i \wedge dx^j = 2 \sum_i (\partial_i G_j - \partial_j G_i) dx^i \wedge dx^j = 0$ since $(\partial_i G_j - \partial_j G_i) = 0$ is just another way of expressing $DG^\top - DG = 0$. Therefore by Poincare’s lemma, $G = d\phi$ for some 0-form (scalar function) ϕ . Therefore $F = J^{-1}d\phi = Jd(-\phi)$ since $J^{-1} = -J$. As the exterior derivative of scalar function is just the gradient, we can define $H = -\phi$ and see that there exists a scalar function H such that $F = J\nabla H$.

A.3 MUJOCO EXPERIMENT DETAILS

Data collection: for each control task we trained a standard soft actor-critic RL agent to convergence [94]. Note that we had to use modified versions of the Gym environments since the standard environments preprocess observations in ad-hoc ways. For example, Hopper clips the velocity observations to $[-10, 10]^d$ and truncates part of the position.¹ Our versions of the environments simply return $[q, v]$ as the observation. Then we randomly split the episodes in the replay buffer into train and test. The training data was 40K 3-step trajectories (i.e. two transitions) randomly sampled from the training episodes. The test data was 200

¹<https://github.com/openai/gym/blob/master/gym/envs/mujoco/hopper.py#L31>

200-step trajectories randomly sampled from the test episodes. This data-collection strategy is important to the experiment because random controls typically do not cause the agent to cover the entire state-action space. Similarly many control policies are highly cyclical, so it is important to separate train and test splits at the episode level.

Training: we trained each model for 256 epochs using Adam with a batch size of 200 and weight decay ($\lambda = 1e-4$). We used a cosine annealing learning rate schedule, with $\eta_{\max} = 2e-4$, $\eta_{\min} = 1e-6$.

Model Architecture Each network was parameterized as a 2-layer MLP with 128 hidden units. Each model used the Euler integration rule with 8 integration steps per transition step. The step size was determined by the integration step size of the underlying environment, $h = \Delta t/8$.

NODE + SO: given the state z and controls u , a standard NODE takes $dz/dt = f(z, u, \theta)$. However, if $z = [q, v]$ (that is, if both position and velocity are observed), then we already have a good estimate of dq/dt in the observation itself, namely v . Hence we propose only using the network to model acceleration $dv/dt = f(z, u, \theta)$, and to model dq/dt implicitly. It is important to note that we cannot take $dq/dt = v$ because v is observed before the control u is applied. Instead we take $dq/dt = v/2 + (v + h \times dv/dt)/2$, averaging the velocity at time t (before the control is applied) and the predicted velocity at time $t + 1$ (after the control is applied), given an Euler integration step of size h on v .

This integration rule can be viewed as an approximate RK2 step on q_t , where v_{t+1} is approximated via a learned Euler step on v_t . This approach has two benefits. In the first place it constrains the predicted velocity and acceleration to be consistent across time. Second the model is able to take an approximate RK2 step on q at the cost of a single forward pass (instead of 2). The latter is important because integration error can accumulate over long rollouts, even if the model fits the dynamics very well.

A.4 ADDITIONAL EXPERIMENTAL RESULTS

A.4.1 COMPARISON OF LOSS FUNCTIONS

In the experimental results presented in the body of the paper were obtained training on l2 loss between integrated and ground truth trajectories. As noted in feedback the an early version of the paper, this practice goes contrary to prior work using l1 loss for stability [69]. In Figure A.1 we show the result of changing the loss.

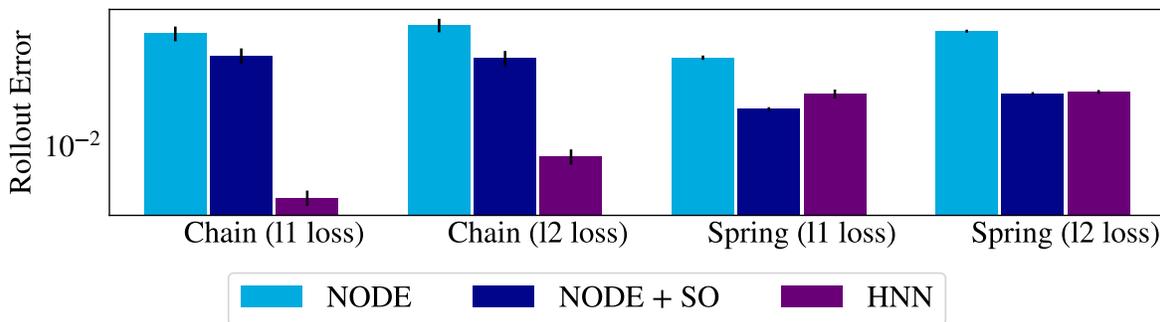


Figure A.1: Switching from l2 to l1 loss can improve rollout error slightly, but doesn't impact the ordering of the models. The other elements of the experimental setup are identical to above. Error bars show one standard deviation.

A.4.2 ADDITIONAL SYSTEMS

To extend the comparison of NODE and HNN models, we trained models on three additional systems presented by Finzi, Wang, and Wilson [69]. Figure A.2 shows the rollout error of NODE, NODE + SO, and HNN models. On the gyroscope system, we observe a similar result to the one above. In the magnet pendulum and rotor systems, the results are slightly more counterintuitive, with the NODE model outperforming the more sophisticated alternatives. We suspect the small difference in performance in the models is due to the challenge of stably training HNNs in complex systems (with magnet pendulum having complex dynamics and rotor having a complex coordinate system).

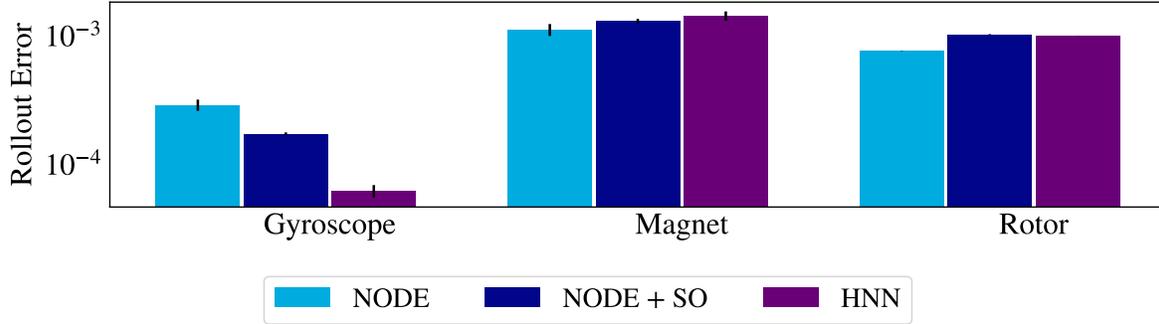


Figure A.2: On the additional systems from Finzi, Wang, and Wilson [69], we can observe the effect of second order structure, compared with NODE and HNN baselines. As before, second order structure seems to account for much of the difference between NODE and HNN models. Error bars show one standard deviation.

A.5 ALIASING EXTENDED DISCUSSION

When working with signal f we can write the fourier series

$$f(k) = \sum_{n=-\infty}^{\infty} F(n)e^{i2\pi kn/N}$$

where $F(n)$ are the frequency components and the values n are called *harmonics*. Take the f to be discretely sampled at uniformly spaced points in a bounded interval $[-1, -1 + \Delta, \dots, 1 - \Delta, 1]$, with $\Delta = 1/K$. Because the sampling rate is limited, it is impossible to correctly measure components $F(n)$ where $n > K/2$. Evaluated only at the grid points, such content could have identical values to components with lower frequencies, causing fundamental ambiguities:

$$\sin(2\pi(k + nK)t + \phi) = \begin{cases} + \sin(2\pi(k + nK)t + \phi) & k + nK \geq 0 \\ - \sin(2\pi|k + nK|t + \phi) & k + nK < 0 \end{cases}$$

The default mechanism for resolving these ambiguities in the reconstruction is to choose the lowest frequency component for the corresponding observations, leading to the aliasing

operation given in [Equation 3.2](#).

This operation can also be considered a translation in the frequency domain. Crucially, operations in frequency domain have corresponding operations in the spatial domain, and thus aliasing can give rise to recognizable patterns in images with poorly chosen resolutions, for example moire patterns. This relationship also means aliasing's effects on translations in frequency space, for example, can effect translational spatial symmetries.

To make this relationship explicit, let us consider the translational symmetries of the first set of feature maps in a CNN in two scenarios. In both scenarios the transformation is downward translation of the input by 10% of its height. First, let us consider the case where this transformation happens to result in a translation by a discrete numbers of pixels, p in the feature maps. Obviously the pixels at the bottom of the image become lost to the boundary and thus cannot be recovered from the corresponding feature maps, as would be required for equivariance, as illustrated in [Figure 3.2](#). As the amount of this translation gets smaller and smaller, however, the effect of the boundary should decrease, and yet the ability to recover the image can still be strongly affected by innate signal processing properties.

Consider the case where the CNN has a stride of 2. The feature maps will have half the width of the original image. Therefore the Nyquist frequency will also be half that of the Nyquist frequency of the image, and there will be aliasing of all the frequencies in between the original Nyquist frequency and the new value. When we try to reverse the transformation by translating p pixels upwards, the resulting translation will no longer be the inverse of the translation on the image. Therefore we cannot achieve perfect equivariance.

As another important subcase, let's also consider the non-linear activation in the CNN layer by itself. If we apply the non-linearity to a translated input, we can simply use the fact the result was a discrete translation in the output space to map the values at the grid points to values at different grid points under the reverse transformation. In this case there is clearly no issue introduced from the frequency domain properties of non-linearities on their

own.

Now let's consider a translation of $1/p$ pixels. In this case, reconstructing the image after the translation is non-trivial, and we need to perform interpolation to calculate the values of the corresponding continuous image at the points that will become translated to the evaluation points. In order to perform this interpolation we must actually consider the full frequency spectrum of the image. Now the effects of pointwise non-linearities can become apparent. Because non-linearities can introduce high frequency content, these high frequencies become important when reconstructing the signal using interpolation. Aliasing makes this reconstruction fundamentally challenging and thus equivariance is impossible to achieve.

A.6 LIE GROUPS, LIE DERIVATIVES, AND LEE

A.6.1 LIE GROUPS AND LOCAL/GLOBAL NOTIONS OF EQUIVARIANCE

The key to understanding why the local - global equivalence holds is that $(\exp(X) - 1) = \sum_{k=1}^{\infty} X^k/k!$ has the same nullspace as X (here repeated application of X on a function f is just the repeated directional derivative, and this is the definition of a vector field used in differential geometry). Since they have the same nullspace, the space of functions for which $\exp(X)f = f$ is the same as the space $Xf = 0$. The same principle holds for $\rho(\exp(X))f = f$ and $d\rho(X)f = 0$ since $\rho(\exp(X)) = \exp(d\rho(X))$ (a basic result in representation theory, which can be found in [96]) where $d\rho$ is the corresponding Lie algebra representation of ρ , which for vector fields is the Lie derivative $d\rho(X) = L_X$. Hence carrying over the constraint for each element $\forall X \in \mathfrak{g} : L_X f = 0$ is equivalent to $\forall X \in \mathfrak{g} : \rho(\exp(X))f = f$ which is the same as $\forall g \in G : \rho(g)f = f$. Unpacking the representation ρ_{12} of f , this is just the global equivariance constraint $\forall g \in G : \rho_2(g)^{-1}f(\rho_1(g)x) = f(x)$.

A.6.2 LIE DERIVATIVE CHAIN RULE

Suppose we have two functions $h : V_1 \rightarrow V_2$ and $f : V_2 \rightarrow V_3$, and corresponding representations ρ_1, ρ_2, ρ_3 for the vector spaces V_1, V_2, V_3 . Expanding out the definition of ρ_{31} ,

$$\begin{aligned}\rho_{31}(g)[f \circ h](x) &= \rho_3(g)^{-1} f(h(\rho_1(g)x)) \\ &= \rho_3(g)^{-1} f(\rho_2(g)\rho_2(g)^{-1}h(\rho_1(g)x)) \\ &= \rho_{32}(g)[f] \circ \rho_{21}(g)[h](x).\end{aligned}$$

From the definition of the Lie derivative, and using the chain rule that holds for the derivative with respect to the scalar t , and noting that $g_0 = \text{Id}$ so $\rho(g_0) = \text{Id}$, we have

$$\begin{aligned}\mathcal{L}_X(f \circ h)(x) &= \left. \frac{d}{dt} \left(\rho_{31}(g_t)[f \circ h](x) \right) \right|_0 \\ &= \left. \frac{d}{dt} \left(\rho_{32}(g_t)[f] \circ \rho_{21}(g_t)[h](x) \right) \right|_0 \\ &= \left(\left. \frac{d}{dt} \rho_{32}(g_t)[f] \right|_{t=0} \right) \circ \rho_{21}(g_0)[h](x) + \left[\left. d(\rho_{32}(g_0)[f]) \right|_{h(x)} \right] \left(\left. \frac{d}{dt} \rho_{21}(g_t)[h] \right|_{t=0} \right)(x) \\ &= \left(\left. \frac{d}{dt} \rho_{32}(g_t)[f] \right|_{t=0} \right) \circ h(x) + df|_{h(x)} \left(\left. \frac{d}{dt} \rho_{21}(g_t)[h] \right|_{t=0} \right)(x) \\ &= (\mathcal{L}_X f) \circ h(x) + df|_{h(x)}(\mathcal{L}_X h)(x),\end{aligned}$$

where $df|_{h(x)}$ is the Jacobian of f at $h(x)$ and $df|_{h(x)}(\mathcal{L}_X h)(x)$ is understood to be the Jacobian vector product of $df|_{h(x)}$ with $(\mathcal{L}_X h)(x)$, equivalent to the directional derivative of f along $(\mathcal{L}_X h)(x)$. Therefore the Lie derivative satisfies a chain rule

A.6.3 STOCHASTIC TRACE ESTIMATOR FOR LAYERWISE METRIC

Unrolling this chain rule for a sequence of layers $\text{NN}(x) = f_{N:1}(x) := f_N(f_{N-1}(\dots(f_1(x))))$, or even an autograd DAG, we can identify the contribution that each layer f_i makes to the

equivariance error of the whole as the sum of terms $C_i = df_{N:i+1} \mathcal{L}_X f_i$, $\mathcal{L}_X(\text{NN}) = \sum_{i=1}^N C_i$.

Each of these C_i , like $\mathcal{L}_X(\text{NN})$ measure the equivariance error for all of the outputs (which we define to be the softmax probabilities), and are hence vectors of size K where K is the number of classes. In order to summarize the C_i as a single number for plotting, we compute their norm $\|C_i\|$ which satisfy $\|\mathcal{L}_X(\text{NN})\| \leq \sum_i \|C_i\|$.

To compute $df_{N:i+1} \mathcal{L}_X f_i$, one can use autograd to perform Jacobian vector products (as opposed to typical vector Jacobian products) and build up $df_{N:i+1}$ in a backwards pass. Unfortunately doing so is quite cumbersome in the PyTorch framework where the large number of available models are implemented and pretrained. A trick which can be used to speed up this computation is to use stochastic trace estimation [14]. Since vector Jacobian products are cheap and easy, we can compute $\|C_i\|^2 = \mathbb{E}[\hat{A}]$ as the expectation of the estimator $\hat{A} = (1/N) \sum_n (z_n^\top C_i)^2 = (1/N) \sum_n (z_n^\top df_{N:i+1} \mathcal{L}_X f_i)^2$ with iid. Normal probe vectors $z_n \sim \mathcal{N}(0, I)$, and the quantity $z_n^\top df_{N:i+1}$ which is a standard vector Jacobian product.

One can see that $\mathbb{E}[\hat{A}] = C_i^\top \mathbb{E}[zz^\top] C_i = C_i^\top I C_i = \|C_i\|^2$. We can then measure the variance of this estimator to control for the error and increase N until this error is at an acceptable tolerance (we use $N = 100$ probes). The convergence of this trace estimator is shown in Figure A.4 (right) for several different layers of a ResNet-50. In producing the final layerwise attribution plots, we average the computed quantity $\|C_i\|$ over 20 images from the ImageNet test set.

A.7 LEE THEOREMS

A.7.1 LEE AND CONSISTENCY REGULARIZATION

As shown in Athiwaratkun et al. [13], consistency regularization with Gaussian input perturbations can be viewed as an estimator for the norm of the Jacobian of the network,

but in fact when the perturbations are not Gaussian but from small spatial transformations, consistency regularization actually penalizes the Lie derivative norm. In the Π -model [131] (the most basic form of consistency regularization), the consistency regularization minimizes the norm of the difference of the outputs of the network when two randomly sampled transformations T^a and T^b are applied to the input,

$$L_{\text{cons}} = \|f(T^a(x)) - f(T^b(x))\|^2. \quad (\text{A.2})$$

Suppose that the two transformations are representations of a given symmetry group and can be written as $T^a = \rho(g_a)$ and $T^b = \rho(g_b)$, and the group elements can be expressed as the flow generated by a linear combination of the vector fields which form the Lie Algebra: $g_a = \Phi_{\sum_i a_i X_i}$ for some coefficients $\{a_i\}_{i=1}^d$ and likewise for g_b . We can define the log map, mapping group elements to their generator values in this basis: $\log(g_a) = a$. Then, assuming a_i are small (and therefore the transformations are small), Taylor expansion yields $L_{\text{cons}} = \|f(x) + \sum_i a_i \mathcal{L}_{X_i} f(x) + O(a^2) - [f(x) + \sum_j b_j \mathcal{L}_{X_j} f(x) + O(b^2)]\|^2$. Therefore, taking the expectation over the distribution which a and b are sampled over (which is assumed to be centered with $\mathbb{E}[a_i] = \mathbb{E}[b_i] = 0$ as well as the input distribution x , we get that

$$\mathbb{E}_{a,b,x}[L_{\text{cons}}] = 2\mathbb{E}[\|\sum_i \mathcal{L}_{X_i} f(x)\|_{\Sigma}^2] + \text{higher order terms}, \quad (\text{A.3})$$

where $\|\cdot\|_{\Sigma}^2$ denotes the norm with respect to the covariance matrix $\Sigma = \text{Cov}(a) = \text{Cov}(b)$.

When the transformations are not parameter space perturbations such as dropout, but input space perturbations like translations (which have been found to be far more important to the overall performance of the method [13]), we can show that consistency regularization coincides with minimizing the expected Lie derivative norm. In this sense, consistency regularization can be viewed as an intervention for reducing the equivariance error on unlabeled data.

A.7.2 TRANSLATION LEE AND ALIASING

Below we show that spatial aliasing directly introduces translation equivariance error as measured by the Lie derivative, where the aliasing operation $A[\cdot]$ is given by [Equation 3.2](#). The Fourier series representation of an image $h(x, y)$ with pixel locations (x, y) is H_{nm} with spatial frequencies (n, m) , where the band limited reconstruction

$$h(x, y) = \frac{1}{2\pi} \sum_{nm} H_{nm} e^{2\pi i(xn+ym)} = F^{-1}[H]$$

and F^{-1} is the inverse Fourier transform, and the sums range over frequencies of $-M/2$ to $+M/2$ for both n and m where M is the image height and width (assumed to be square for convenience).

Applying a continuous translation by $t\mathbf{v}$ along vector $\mathbf{v} = (v_x, v_y)$ to the input means resampling the translated band limited continuous reconstruction $h(x, y)$ at the grid points.

$$T_{t\mathbf{v}}[h](x, y) = h(x - tv_x, y - tv_y) = \frac{1}{2\pi} \sum_{n,m=-M/2}^{M/2} H_{nm} e^{2\pi i[(x-tv_x)n+(y-tv_y)m]}$$

To simplify the notation, we will consider translations along only x and suppress the m index of H_{nm} , effectively deriving the result for the translations of a 1d sequence, but that extends straightforwardly to the 2 dimensional case.

$$T_{t\mathbf{v}}[h](x) = h(x - tv_x) = \frac{1}{2\pi} \sum_{n=-M/2}^{M/2} [H_n e^{-2\pi i tv_x n}] e^{2\pi i x n}$$

Applying the aliasing operation, sampling the image to a new size M' (with Nyquist

frequency $M'/2$), we have

$$\begin{aligned} A[T_{t\mathbf{v}}[h]](x) &= \frac{1}{2\pi} \sum_{n=-M/2}^{M/2} [H_n e^{-2\pi i t v_x n}] e^{2\pi i x \text{Alias}(n)} \\ &= \frac{1}{2\pi} \sum_{n'=-M'/2}^{M'/2} \left[\sum_{n=\text{Alias}^{-1}(n')} H_n e^{-2\pi i t v_x n} \right] e^{2\pi i x n'} \end{aligned}$$

where the last line follows from applying a change of variables $n' = \text{Alias}(n)$.

Applying the final inverse translation (which acts on the M' sampling rate band limited continuous reconstruction), we have

$$T_{-t\mathbf{v}}[A[T_{t\mathbf{v}}[h]]](x) = \frac{1}{2\pi} \sum_{n'=-M'/2}^{M'/2} \left[\sum_{n=\text{Alias}^{-1}(n')} H_n e^{-2\pi i t v_x (n-n')} \right] e^{2\pi i x n'}.$$

Taking the derivative with respect to t , we have

$$\begin{aligned} \mathcal{L}_{\mathbf{v}}(A)(h) &= \left. \frac{d}{dt} \right|_0 T_{-t\mathbf{v}}[A[T_{t\mathbf{v}}[h]]] \\ &= \frac{1}{2\pi} \sum_{n'=-M'/2}^{M'/2} \left[\sum_{n=\text{Alias}^{-1}(n')} 2\pi i v_x (n' - n) H_n \right] e^{2\pi i x n'}. \end{aligned}$$

Notably, for aliasing when the frequency is reduced by a factor of 2 from downsampling, there are only two values of n that satisfy $\text{Alias}(n) = n'$: the value $n = n'$ and the one that gets aliased down, therefore when multiplied by $n - n'$ the sum

$$\left[\sum_{n=\text{Alias}^{-1}(n')} 2\pi i v_x (n' - n) H_n \right]$$

consists only of a single term.

According to Parseval's theorem, the Fourier transform F is unitary, and therefore the norm of the function as a vector evaluated at the discrete sampling points $x = 1/M', 2/M', \dots$

is the same as as the norm of the Fourier transform:

$$\begin{aligned} \|\mathcal{L}_{\mathbf{v}}(A)(h)\|^2 &= \|F[\mathcal{L}_{\mathbf{v}}(A)(h)]\|^2 \\ \|\mathcal{L}_{\mathbf{v}}(A)(h)\|^2 &= \sum_{n'=-M'/2}^{M'/2} \left| \sum_{n=\text{Alias}^{-1}(n')} 2\pi i v_x(n' - n) H_n \right|^2 \\ \|\mathcal{L}_{\mathbf{v}}(A)(h)\|^2 &= \sum_{n=-M/2}^{M/2} (2\pi)^2 v_x^2(\text{Alias}(n) - n)^2 H_n^2, \end{aligned}$$

using the fact that only one element is nonzero in the sum. Finally, generalizing to the 2d case, we have

$$\|\mathcal{L}_{\mathbf{v}}(A)(h)\|^2 = (2\pi)^2 \sum_{nm} H_{nm}^2 \left(v_x^2(\text{Alias}(n) - n)^2 + v_y^2(\text{Alias}(m) - m)^2 \right), \quad (\text{A.4})$$

showing how the translation Lie derivative norm is determined by the higher frequency components which are aliased down.

A.8 LEARNED EQUIVARIANCE EXPERIMENTS

A.8.1 LAYER-WISE EQUIVARIANCE BASELINES

We use EQ-T and EQ-T_{frac} [126] to calculate layer-wise equivariance by caching intermediate representations from the forward pass of the model. For image-shaped intermediate representations, EQ-T samples integer translations in pixels between -12.5% and 12.5% of the image dimensions in pixels. EQ-T_{frac} is identical but with continuous translation vectors. The individual layer is applied to the transformed input and then the inverse group action is applied to the output, which is compared with the original cached output. Many different normalization could be chosen to compare equivariance errors across layers. The most obvious are $\frac{1}{N}$, $\frac{1}{\sqrt{N}}$, and $\frac{1}{1}$ (no normalization), where $N = C \times H \times W$. The normalization

method can have a large effect of the relative contribution of a layer, despite the decision being relatively arbitrary (in contrast to LEE, which removes the need for doing so as the scale is automatically measured relative to the contribution to the output).

A.8.2 SUBNETWORK EQUIVARIANCE ANALYSIS

Another way one might use LEE to study the effects of different layers that make up a network is to break the network in question down into its constituent subnetworks (networks starting at the input and ending at every intermediate representation in the network) and calculate the LEE of the corresponding function. We show the result of this calculation on a ResNet50 in [Figure A.3](#).

As an alternative to our layerwise analysis, this method has a significant drawback that makes analysis challenging: the functions under consideration have different outputs. In our calculation, we applied batch normalization over the outputs in order to make their scales comparable. Despite this rescaling, comparing activations and preactivations, for example, remains challenging. By contrast, our layerwise breakdown specifically targets a layer’s contribution to a shared output.

A.8.3 MODEL LIST

The models included in Figure 1 are

- Early CNNs: ResNets [101], ResNeXts [231], VGG [192], Inception [197], Xception [45], DenseNet [114], MobileNet [186], Blur-Pool Resnets and Densenets [245], ResNeXt-IG [152], SeResNe*ts [113], ResNet-D [103], Gluon ResNets [91, 243, 247], SKResNets [142], DPNs [43]
- Modern CNNs: EfficientNet [198, 199], ConvMixer [212], RegNets [177], ResNet-RS, [19], ResNets with new training recipes [225], ResNeSts [243], RexNet [97], Res2Net

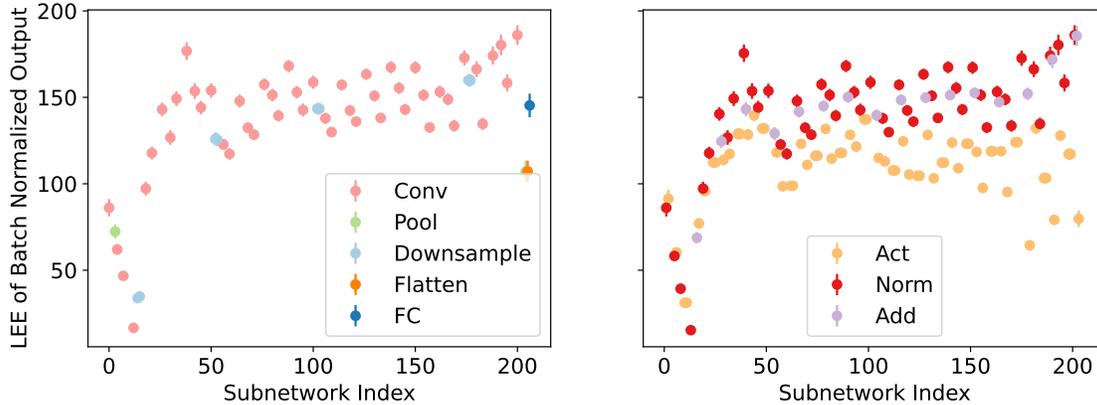


Figure A.3: LEE calculated over the subnetworks of a ResNet50. Specifically a subnetwork is constructed between the input and every intermediate representation in the network’s computation graph. We use batch normalization of the outputs to make the output scale of different subnetwork comparable. For visual clarity, layer types are broken across the left and right plots, which share the same axes. Similar to the pattern observed in Figure 3.4, we see a rapid increase in equivariance error in the early layers of the network, followed by many smaller increases later in the network. Unlike in our layerwise decomposition, comparison across layer types is challenging in this setting because layers have significantly different outputs. For example, comparing activations with preactivations is complicated by the ReLUs acting as contractions of the input and having potentially many zeroed values.

[77], RepVGG [62], NFNet [27], XNect [156], MixNets [200], ResNeXts with SSL pretraining [235], DLA [237], CSPNets [216], ECA NFNet and ResNets [27], HRNet [194], MnasNet [201]

- Vision transformers: ViT [63], CoaT [56], SwinViT [148], [17], CaiT [205], ConViT [54], CrossViT [38], TwinsViT [47], TnT [98], XCiT [163], PiT [106], Nested Transformers [248]
- MLP-based architectures: MLP-Mixer [210], ResMLP [209], gMLP [145], MLP-Mixers with (Si)GLU [224]

A.8.4 ALTERNATIVE END-TO-END EQUIVARIANCE METRICS

DISCRETE CONSISTENCY We adopt the consistency metric from Zhang [245], which simply measures the fraction of top-1 predictions that match after applying an integer translation

to the input (in our case by 10 pixels). Instead of reporting consistency numbers, we report $(1 - \%$ matching), so that consistency becomes a measure of equivariance error. Equivariant models should exhibit end-to-end invariance, high consistency, and low equivariance error.

EXPECTED GROUP SAMPLE EQUIVARIANCE Inspired by work in equivariant architecture design [71, 116], we provide an additional equivariance metric for comparison against the Lie derivative. Following [116], we sample k group elements in the neighborhood of the identity group element, with sampling distribution $\mathcal{D}(G)$, and calculate the sample equivariance error for model f as $\frac{1}{k} \|\rho_2^{-1}(g)f(\rho_1(g)x) - f(x)\|$. For translations we take $\mathcal{D}(G)$ to be $\text{Uniform}(-5, 5)$ in pixels.

VERSUS LEE There are several reasons why the continuous lie derivative metric is preferable over discrete and group sample metrics. Firstly, it allows us to break down the equivariance error layerwise enabling more fine grained analysis in a way not possible with the discrete analog. Secondly, the metric is less dependent on architectural details like the input resolution of the network. For example, for discrete translations by 1 pixel, these translations have a different meaning depending on the resolution of the input, whereas our lie derivatives are defined as the derivative of translations as a fraction of the input size, which is consistently defined regardless of the resolution. Working with the vector space forming the Lie algebra rather than the group also removes some unnecessary freedom in how one constructs the metric. Rather than having to choose an arbitrary distribution over group elements, if we compute the Lie derivatives for a set of basis vectors of the lie algebra, we have completely characterized the space, and all lie derivatives are simply linear combinations of the computed values. Finally, paying attention to continuous transformations reveals the problems caused by aliasing which are far less apparent when considering discrete transformations, and ultimately the relevant transformations are continuous and we should study them directly.

A.8.5 LEE FOR ADDITIONAL TRANSFORMATIONS

Beyond the 3 continuous transformations that we study with Lie derivatives above, there are many more that might reveal important properties of the network. Here we include an three additional transformations—hyperbolic rotation, brightening, and stretch.

Figure A.4 (left) shows that, perhaps surprisingly, models with high accuracy become more equivariant to hyperbolic rotations. We suspect this surprisingly general equivariance to diverse set of continuous transformations is probably the result of improved anti-aliasing learned implicitly by more accurate models. LEE does not identify any significant correlation between brightening or stretch transformations and generalization ability.

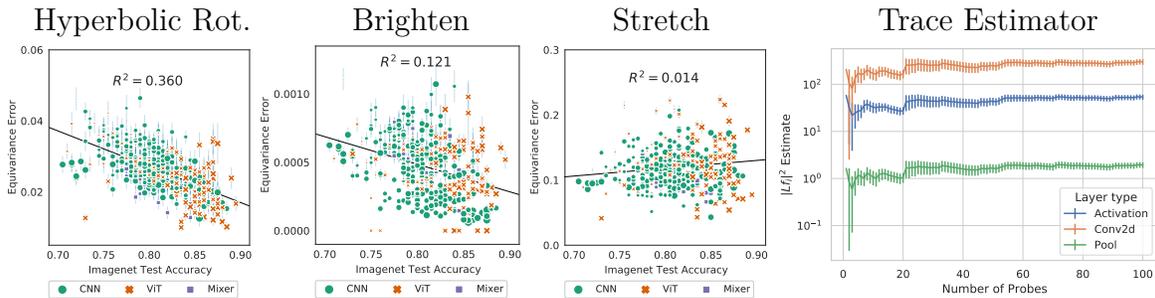


Figure A.4: (Left): Extending Figure 3.5 we show the Lie derivative norm for hyperbolic rotation, brightening, and stretch transformations. We observe that more accurate models are also more equivariant to hyperbolic rotations and to brighten transformation, to a more limited extent. In the case of hyperbolic rotations, this result is surprising, as nothing has directly encouraged this equivariance. One possible explanation is decreased aliasing in models with higher accuracy. Marker size indicates model size. Error bars show one standard error over the images used to evaluate the Lie derivative. **(Right):** Cumulative mean and standard error of the estimator (computed for translations on a ResNet-50).

A.8.6 ROTATED MNIST FINETUNING

In order to test the ability of SOTA imagenet pre-trained models to learn equivariance competitive with specialized architectures, we adapted the example rotated MNIST notebook available in E2CNN repository [222]. We use the base model and default finetuning procedure from [102], finetuning for 100 epochs, halving the learning rate on loss plateaus.

B | APPENDIX: TRANSFER FROM TEXT TO NUMERICAL DATA

B.1 DETAILED METHOD AND HYPERPARAMETERS

B.1.1 INPUT SCALING

For all baseline methods, we use the `MinMaxScaler` from `sklearn`. For GPT-3, since it can handle inputs spanning multiple orders of magnitudes by using varying number of digits, we apply an affine transformation to each element x_t of a time series (x_1, \dots, x_T) : $x_t \mapsto (x_t - b)/a$, where $b = \min_t x_t - \beta(\max_t x_t - \min_t x_t)$, and a is the α -percentile of the shifted series $(x_1 - b, \dots, x_T - b)$. We also consider a basic scaler that only applies scaling and not shifting, with a clipped to a maximum of 0.01 when the series only has tiny values. Here α and β are hyperparameters controlling the thresholds at which the number of digits used by the language model changes.

B.1.2 VALIDATION TUNING

We construct a validation time series from the last T observations in the training series, where T is the length of the test series. When the training series is shorter than $2T$, we take the last half of the training series as the validation series. The likelihood of generating the

validation conditioned on the remaining training series is used to select the hyperparameters. Since `LLMTime` is zero-shot, the likelihood is computed without training. For other methods such as ARIMA, the likelihood is computed after training on the remaining training series.

B.1.3 LIKELIHOOD ADJUSTMENT FOR GPT MODELS

In order to convert token probabilities assigned by language models into continuous densities, we must convert the distribution over all possible tokens into the distribution over only tokens used in the numerical encoding scheme. When we have access to the language model’s logits, performing this adjustment is easy. We can simply set the probability of any non-essential tokens to zero and renormalize the distribution. When using black-box APIs (e.g. the OpenAI API), however, we need to approximate this procedure, because it frequently impossible to obtain the full discrete distribution over tokens. For example, in the OpenAI API, only the top 5 log probabilities are returned for every step in the generation process. As we cannot properly renormalize the distribution, we have to make adjustments that get us as close as possible to the true renormalized distribution, as we show in the following calculations. We take p to be the raw probabilities assigned by a language model and \tilde{p} to be the adjusted probabilities, with

$$p_k = \frac{e^{\log p_k}}{\sum_i e^{\log p_i}} , \quad \tilde{p}_k = \frac{e^{\log p_k}}{(\sum_i e^{\log p_i}) - e^{l_0}}$$

where l_0 is the probability that mass assigned to tokens that are not part of the numerical encoding scheme. When access to log probabilities is limited, l_0 can be approximated as the sum of the log probabilities for non-numerical tokens in the top k . From the definition of \tilde{p}

we can derive

$$\begin{aligned} \frac{1}{\tilde{p}} &= e^{l_0} + \frac{1}{p} \implies \\ \tilde{p} &= \left(\frac{1}{p} - e^{l_0}\right)^{-1} \implies \\ \log \tilde{p} &= \log p - \log(1 - e^{l_0}) \end{aligned}$$

B.2 BENCHMARKING DETAILS AND EXTENDED RESULTS

B.2.1 DARTS DATASETS

For the Darts datasets, we use the GPyTorch library [78] for Gaussian Process implementation and the Darts library [107] for ARIMA, TCN, N-BEATS, N-HiTS. We use default values for hyperparameters not described below. The test set is the last 20% of each series.

We use several baseline methods implemented directly in Darts [107]:

- **ARIMA:** ARIMA [24], short for AutoRegressive Integrated Moving Average, has been a popular choice for time series forecasting for many decades. ARIMA in Darts wraps code from [79].
- **TCN:** Temporal Convolutional Network (TCN) [134] is residual network with dilated 1D convolutions.
- **N-BEATS:** N-BEATS [170] is a deep learning model tailored for time series forecasting. It employs a deep architecture with backward and forward residual links and stacked fully-connected layers.
- **N-HiTS:** N-HiTS [33] is a deep learning model that incorporates hierarchical interpolation and multi-rate data sampling techniques in order to create forecasts that emphasize different frequencies and scales of the input signal.

We also include Spectral Mixture Gaussian Process (**SM-GP**) [226] as a Bayesian nonparametric approach to time series modeling.

We include the exact hyperparameters for each method below:

GPT-3 We perform a grid search over $\alpha \in [0.5, .7, 0.9, 0.99]$, $\beta \in [0, .15, 0.3, .5]$, precision (number of decimals) $\in [2, 3]$, and temperature = 0.7.

GPT-4 Since likelihood evaluation is not available for GPT-4, we fix its hyperparameters for all datasets as follows: we use the basic scaler with $\alpha = 0.3$ and temperature = 1.0 with top p = 0.8. We do not insert spaces between digits for GPT-4 since it uses a different tokenizer than GPT-3 for which this strategy is not effective.

LLAMA For models LLaMA-1 (7B/13B/30B/70B) and LLaMA-2 (7B/7B-chat/13B/13B-chat), we perform a grid search over temperature $\in [0.2, 0.4, 0.6, 0.8]$ and use $\alpha = 0.99$, $\beta = 0.3$, precision = 3, nucleus = 0.9. For LLaMA-2 70B and LLaMA-2 70B-chat we use temperature = 1.0, $\alpha = 0.99$, $\beta = 0.3$, precision = 3, nucleus = 0.9.

SPECTRAL MIXTURE GAUSSIAN PROCESS (SM-GP) We use a GP with a kernel formed by the sum of a spectral mixture kernel with 12 mixture components and a RBF kernel. We tune the learning rate from [5e-3, 1e-2, 5e-2, 1e-1].

ARIMA We perform a grid search over $p \in [12, 20, 30]$, $d \in [1, 2]$, and $q \in [0, 1, 2]$.

TCN We perform a grid search over

- input_chunk_length $\in [10, 100, 400]$
- output_chunk_length $\in [1, 10]$
- kernel_size $\in [3, 5]$

- `num_filters` $\in [1, 3]$
- `likelihood` $\in [\text{Laplace}, \text{Gaussian}]$

N-BEATS We perform a grid search over

- `input_chunk_length` $\in [10, 100, 400]$
- `output_chunk_length` $\in [1, 10]$
- `layer_widths` $\in [64, 16]$
- `num_layers` $\in [1, 2]$
- `likelihood` $\in [\text{Laplace}, \text{Gaussian}]$

N-HITS We perform a grid search over

- `input_chunk_length` $\in [10, 100, 400]$
- `output_chunk_length` $\in [1, 10]$
- `layer_widths` $\in [64, 16]$
- `num_layers` $\in [1, 2]$
- `likelihood` $\in [\text{Laplace}, \text{Gaussian}]$

B.2.2 MONASH DATASETS

We evaluate on 19 datasets in Monash that satisfy two criteria:

1. The total number of individual series cannot be prohibitively large, so that the experiments can be run in time without access to an enormous cluster and without a gratuitous API expenses.

2. The length of the forecasting horizon cannot extend to a length that makes it impossible to fit both the forecast and the history into the context window of the language model.

When we applied these criteria, we obtained the following 19 datasets were selected: covid deaths, solar weekly, tourism monthly, australian electricity demand, pedestrian counts, traffic hourly, hospital, fred md, tourism yearly, tourism quarterly, us births, nn5 weekly, nn5 daily, traffic weekly, saugeenday, cif 2016, bitcoin, sunspot.

To aggregate across datasets, we normalized the mean absolute error by the MAE achieved by simply predicting the last observed value before the test series (a naive baseline). This normalization places high weight on datasets for which methods perform significantly better or worse than the naive predictor.

Several of the baseline methods in the archive are shared with Darts, and all descriptions and code can be found in [81]. A few notable addition include

- **CatBoost**: CatBoost [175] is gradient-boosting framework for continuous or categorical data.
- **FFNN**: A feed-forward neural network with a fixed window of input and output, inspired by Goodfellow, Bengio, and Courville [84].
- **PR**: A linear pooled regression (PR) model proposed by Trapero, Kourentzes, and Fildes [211].

We include visualizations of GPT-3’s prediction on these datasets in Appendix B.2.7.

GPT-3 HYPERPARAMETERS We use the following hyperparameters for GPT-3: $\alpha = 0.9, \beta = 0, \text{temperature} = 0.7$. To avoid exceeding the context window, we truncate the history to at most 500 most recent observations. For the baselines, we report their performance as presented in [81]. The normalized MAE values shown in Figure 4.5 (center) are obtained by normalizing by the lowest baseline MAE on each dataset before aggregating.

LLAMA-2 70B HYPERPARAMETERS We use the following hyperparameters for LLaMA-2 70B: $\alpha = 0.99$, $\beta = 0.3$, temperature = 1.0, nucleus = 0.9. To avoid exceeding the context window, we truncate the history to fit in the LLaMA-2 context window (4096 tokens).

B.2.3 INFORMER DATASETS

There are 6 datasets used by Zhou et al. [254] that have become standard benchmarks for evaluating efficient transformers. We evaluate on the 5 datasets that are typically used with a prediction horizon of 96 or 192: “ETTm2”, “exchange_rate”, “electricity”, “traffic”, and “weather”. The results provided in the main text are for a prediction horizon of 96, and we include results for prediction horizon 192 in Appendix B.2.6. To make evaluation tractable with LLMTIME, we use a smaller evaluation set for each dataset, taking the last 96 or 192 timesteps of each series within each dataset as the test set. As there are many individual series in each multivariate dataset, the number of individual timesteps in the test sets is still substantial. To forecast multivariate series with LLMTIME we simply forecast over each series independently, combine the results, and evaluate as in prior work. Our efficient transformer baselines include

- **Informer:** Informer [254] is an efficient transformer model with sparse attention designed for long sequences.
- **Reformer:** Reformer [254] uses a locality-sensitive hashing mechanism to improve the memory use of attention.
- **Autoformer:** Autoformer [230] is a model design for long time series that replaces standard attention with a mechanism in Fourier space.
- **FEDformer:** Like Autoformer, FEDformer [256] uses frequency-based decompositions to construct an efficient alternative to attention.

LLAMA-2 70B HYPERPARAMETERS We use LLaMA-2 70B with $\alpha = 0.99, \beta = 0.3$, temperature = 1.0, nucleus = 0.9, precision = 3. The series in the Informer datasets are very long and we put as much as possible in the LLaMA-2 context window (4096).

B.2.4 SYNTHETIC DATASETS

For the baselines, we use the same hyperparameter grid in Section B.2.1. For GPT-3, we didn't find it useful to perform validation tuning. We use the basic scaler with $\alpha = 0.1$ and temperature = 0.7.

B.2.5 DARTS FULL PROBABILISTIC RESULTS

In Figure B.1 we show the predicted NLLs and forecasts from LLMTIME using GPT-3 and LLaMA-2 70B as base models. LLMTIME typically obtains much better likelihoods than baselines and successfully identifies trend and seasonal components in the time series. We attribute this strong performance in part to the fact that the time series are relatively short. With the tokenization of the input, only about 300 of the observations can fit into the context window, and among the datasets only `Sunspots` and `HeartRate` exceed this amount (with 705 and 900 observations respectively).

B.2.6 INFORMER DATASETS WITH EXTENDED HORIZON

Figure B.2 shows MAE results per dataset and in aggregate for the Informer datasets we used in the paper. Extending the results in the main text, we also include MAE numbers for a prediction horizon of 192. We observed a similar trend overall, though the relative performance of LLMTIME is slightly diminished, largely due to the “electricity” and “traffic” datasets.

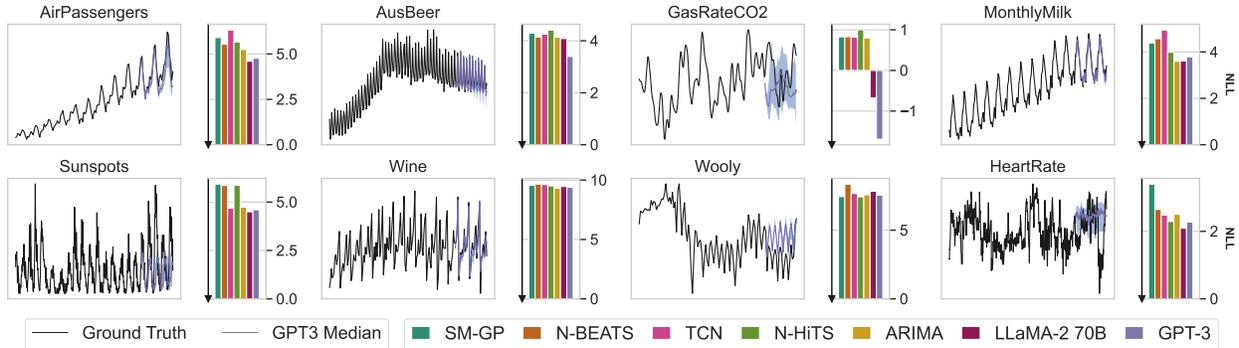


Figure B.1: Median predictions of LLMTime (GPT-3) and NLLs from LLMTime (GPT-3 and LLaMA-2 70B) for every dataset within Darts [107]. The shaded area shows the 10th to 90th quantiles of the distribution over samples. LLMTime consistently obtains better likelihood values than the baselines and often makes surprisingly accurate forecasts by effectively extrapolating trend and periodic components.

B.2.7 MONASH DATASET VISUALIZATIONS

Figure B.3 shows visualizations of the LLMTime’s median predictions (GPT-3 base model) on a subset of the Monash datasets.

B.2.8 INFORMER DATASET VISUALIZATIONS

Figure B.4 shows visualizations of the LLMTime’s median predictions (LLaMA-2 70B base model) on the Informer datasets, for a subset of the each set of multivariate series.

B.3 SIMPLICITY BIAS EXPERIMENTS

We generate data from the function $f(x) = \cos(x) + x$ and add Gaussian noise with zero mean and variance 0.05. We fit symbolic expressions to the first 140 timesteps using PySR [50] with symbols ["+", ".", "-", "/", "sin", "cos", "exp", "square"] and `maxsize = 70`, `maxdepth = 10`, `population_size = 50`, `loss = abs(prediction - target)`, `model_selection = accuracy` and `iterations = 100`. The solutions are saved and ranked by complexity, which is simply the number of terms in the symbolic regression function. The five solutions shown in Figure 4.8

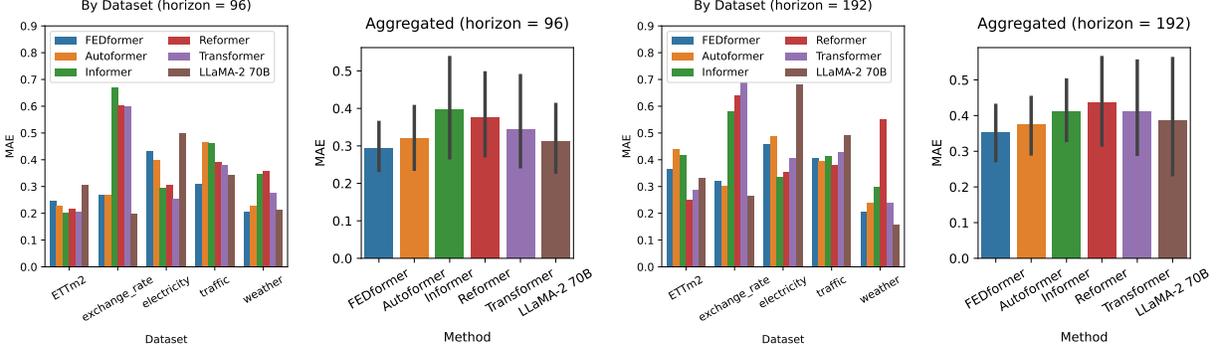


Figure B.2: Aggregated and non-aggregated MAE numbers for LLTime (LLaMA-2 70B base model) and baselines on the Informer datasets. Overall LLTime performs well in aggregate for a zero-shot method, but its performance is highly variable, being the best method on some datasets and the worst on others. The relative performance of LLTime is slightly diminished for a longer prediction horizon, but LLTime is still very competitive with the best methods in aggregate. Error bars show two standard deviations in the error over datasets.

are

1. $(x_0 + 0.3652524)$
2. $\cos(\cos(x_0 / -0.031412385) * (-1.5252972 + x_0))$
3. $(\sin(\cos(\cos(x_0 / 0.031470172) * -1.4668792)) + (\cos(0.81965065) * x_0))$
4. $(\sin(\cos(\cos((x_0 / \sin(-0.03127001)) + 0.07646165) * -1.4539052)) + (\sin(\sin(\cos(\cos(\exp(\cos(-0.03127001) + x_0)))))) * x_0))$
5. $(\cos((\cos((x_0 / -0.03127001) + 0.07646165) / -0.957405) / \sin(\sin(\cos(x_0 - x_0)) * \exp(\cos(\sin(x_0 / -0.983214)))))) / (\cos(\sin(\sin(\sin(\sin(x_0)) - (x_0 * (-0.47036648 - (x_0 / 0.5857117)))))) - -0.10476875))$

To obtain likelihoods we run GPT-3 ('text-davinci-003') with $\alpha = 0.99$, $\beta = 0.3$, $\text{basic} = \text{True}$, $\text{precision} = 1$, $\text{signed} = \text{True}$.



Figure B.3: LLTime (GPT-3 base model) median predictions on at most 4 randomly chosen series per Monash dataset.

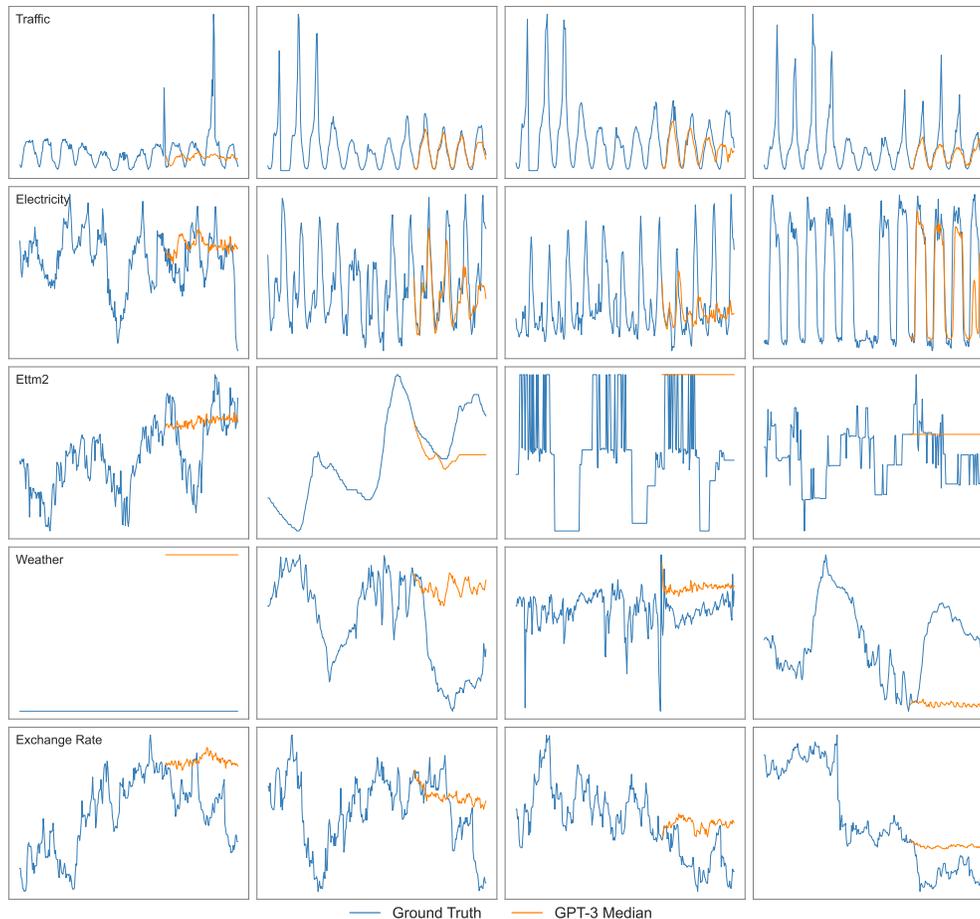


Figure B.4: LLTime (LLaMA-2 70B base model) median predictions on 4 randomly chosen series per Informer dataset.

B.3.1 FULL SYNTHETIC PREDICTIONS

Figure B.5 shows likelihoods and forecasts from LLTime with GPT-3 on the full set of synthetic datasets. We see that some compositional tasks like Linear + Cosine are challenging, while others (Linear * Sine or X * Sine) are well within the abilities of the model. As shown above, GPT-3 demonstrates good understanding of Linear + Cosine through its likelihoods, but has more trouble in sampling. This discrepancy could be the result of good solutions being high likelihood while not being *typical*.

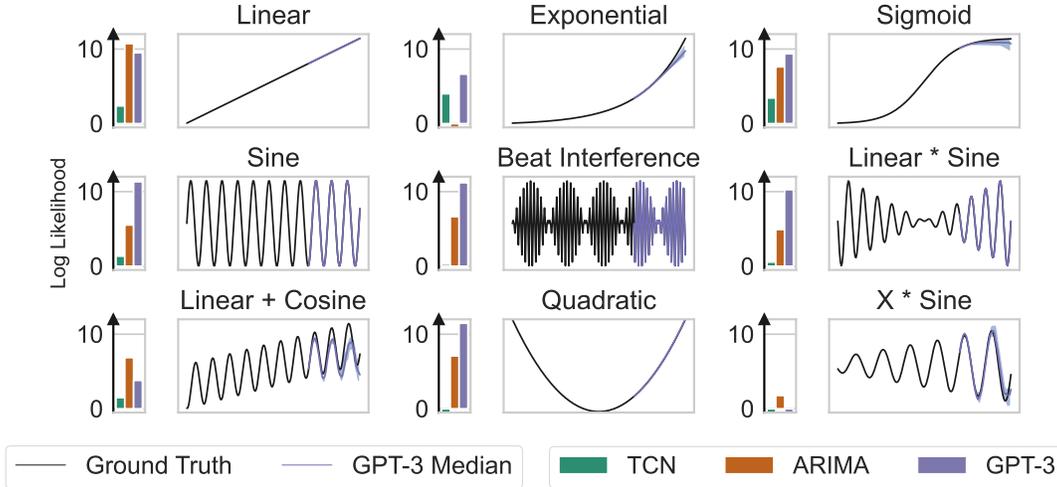


Figure B.5: LLMTime median predictions on all synthetic datasets using GPT-3 as a base model. The hyperparameters used are described in Appendix B.2.4.

B.4 GPT-4

We investigated using GPT-4 for time series prediction. Due to the limitations of the tokenizer, we used the naive tokenization strategy of feeding in the numbers without additional spaces. In addition, due to the enforced separation between system and user in the interface (through additional tokens we cannot modify), inputting the time series input alone leads GPT-4 to talk about the time series or provide analysis, rather than simply continuing the stream of numbers. In order to coax GPT-4 to produce numerical predictions which can be decoded, we added the additional commands System: "You are a helpful assistant that performs time series predictions. The user will provide a sequence and you will predict the remaining sequence. The sequence is represented by decimal strings separated by commas." User: "Please continue the following sequence without producing any additional text. Do not say anything like 'the next terms in the sequence are', just return the numbers. Sequence:". We found that doing so was sufficient to be able to consistently decode the output numerically for GPT-4, but not for GPT-3.5-turbo.

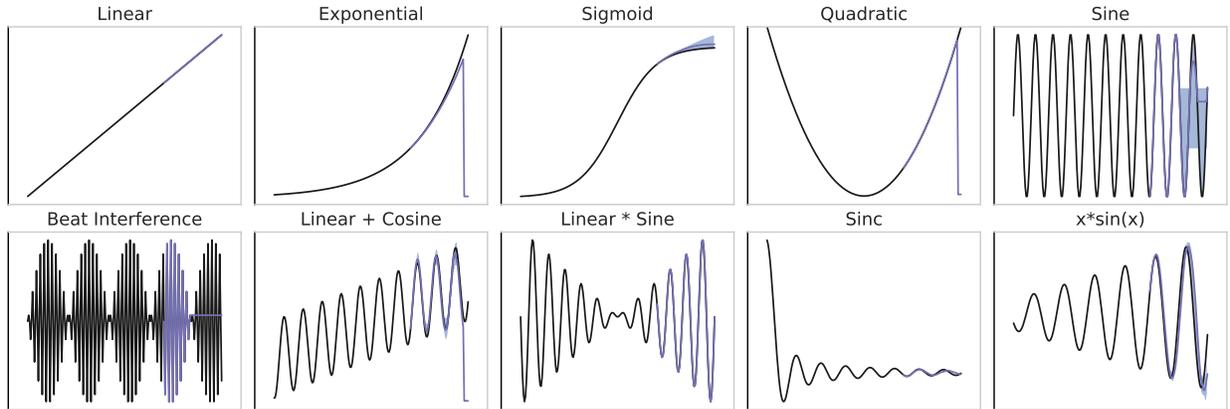


Figure B.6: GPT-4 extrapolations on synthetic data (10-90th percentiles shaded). GPT-4 is able to identify and extrapolate the pattern for each of the deterministic time series, but sometimes behaves erratically.

We show predictions on the synthetic benchmarks (from [Figure 4.8](#)) in [Figure B.6](#). As one can observe, GPT-4 is considerably better performing on these synthetic benchmarks, although numerical decoding of the model sometimes fails before the full output. With non-deterministic time series problems such as with the DARTS datasets, the predictions are slightly worse than GPT-3, but the uncertainties are much less well calibrated as shown in [Figure B.7](#).

B.5 MULTIMODAL TEXT UNDERSTANDING OF TIME SERIES

We evaluate the ability of the language model to reason about the input time series through text in a zero-shot fashion. To test this, we devise a simple experiment where we generate a synthetic time series from one of several candidate functions. We provide the generation code and the numerical values to GPT-4 ([Listing 1](#)), but because of the randomness, GPT-4 must infer which of the functions generated the values. We note that as this code could easily be found within a Jupyter notebook on the internet without intentionally

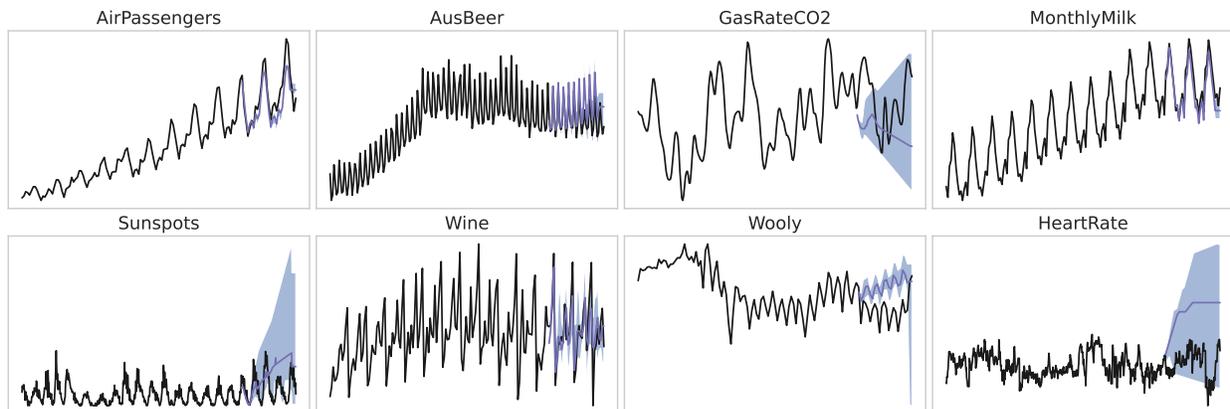


Figure B.7: GPT-4 extrapolations on real (DARTS) time series (10-90th percentiles shaded). The extrapolations are plausible but worse than GPT-3, and the uncertainties tend to be more poorly calibrated making for a high CRPS.

being designed as an experiment for LLMs, we should expect that this textual time series identification task will fall within the data distribution, and in principle should be solved given sufficient capabilities of the language model.

To make the problem slightly easier, we add an additional guiding prompt before and after the text in Listing 1. We prepend

“The following is code that was run to generate a synthetic time series. From the input and output you will be asked to identify which of the time series was picked to generate the data.”

to the code, and after the time series we append either

“Which name gave rise to this series? Put your answer in the form ‘Answer: gaussian_wave’ ”

or

“Carefully analyze the time series. Think step by step, make observations about the time series that you see and then use your observations to identify which of

the functions is most likely to have generated it. Reason your way to a solution and at the end give a name as your answer such as ‘Answer: gaussian_wave.’”

for chain-of-thought prompting.

The prediction accuracies computed over 20 trials are shown in Figure 4.10, with `x_times_sine`, `beat`, and `sinc` not shown in the table because GPT-4 predicted these incorrectly 100% of the time. With the CoT prompting, this prediction task elicits some interesting textual analysis of the time series. Several (non cherry-picked) examples are shown below. Notably, this task elicits the model to analyze the time series in text, reasoning about the trend and periodicity. However, the model sometimes makes incorrect deductions about the behavior of the data it has seen, or the expected behavior of the candidate functions.

Listing B.1: Self-contained code presented to the model for the multimodal time series identification task. When the code is run one of the listed functions is randomly chosen to generate the time series. In order to simply predict the next token after observing this text (which could be found in the cells of a Jupyter notebook) the model must infer which of the functions produced the series.

```
import numpy as np

mapping = {

    'gaussian_wave': lambda t: np.exp(-5*(t-.6)**2)*np.sin(20*(t-6)),
    'exp': lambda t: np.exp(2*t),
    'linear_cos': lambda t: 0.3+ 0.5*t +.2*np.cos(25*t+3),
    'linear': lambda t: 0.3+ 0.5*t,
    'sine': lambda t: np.sin(40*t+3),
    'sinc': lambda t: np.sin(10*t)/t/10,
    'beat': lambda t: np.sin(3*t)*np.sin(25*t),
    'sigmoid': lambda t: 1/(1+np.exp(-4*t)),
    'log': lambda t: np.log(1+t),
    'x_times_sine': lambda t: 4*(t+1)*np.sin(10*(t+1)+4),
    'square': lambda t: 3*(t-.6)**2,
```

```

}
name = np.random.choice(list(mapping.keys()))
t = np.linspace(-1,1,200)+.1*np.random.randn(1)
x = mapping[name](t)
np.set_printoptions(
    formatter={'float': lambda x: "{0:0.3f}".format(x)}
)
print("Series:␣",x)
print("␣",name)

```

Series:

```

[-0.000 -0.033 -0.070 -0.111 -0.153 -0.197 -0.240 -0.281
-0.320 -0.355 -0.385 -0.408 -0.425 -0.433 -0.432 -0.422
-0.402 -0.371 -0.330 -0.279 -0.217 -0.145 -0.064 0.026
 0.124 0.229 0.339 0.453 0.570 0.688 0.806 0.922 1.033
 1.140 1.238 1.328 1.407 1.474 1.527 1.564 1.586 1.590
 1.576 1.543 1.491 1.420 1.329 1.219 1.091 0.945 0.782
 0.604 0.413 0.209 -0.005 -0.227 -0.455 -0.686 -0.917
-1.147 -1.373 -1.591 -1.799 -1.995 -2.175 -2.338 -2.481
-2.602 -2.698 -2.769 -2.812 -2.826 -2.811 -2.766 -2.689
-2.583 -2.446 -2.279 -2.084 -1.863 -1.615 -1.345 -1.053
-0.743 -0.418 -0.081 0.265 0.617 0.970 1.322 1.667 2.003
 2.325 2.630 2.915 3.175 3.407 3.609 3.778 3.910 4.005
 4.060 4.074 4.046 3.975 3.862 3.706 3.509 3.272 2.997
 2.686 2.341 1.965 1.563 1.137 0.692 0.232 -0.239 -0.715
-1.192 -1.665 -2.128 -2.577 -3.008 -3.414 -3.791 -4.136

```

-4.443 -4.710 -4.932 -5.108 -5.233 -5.306 -5.326 -5.291
-5.202 -5.058 -4.859 -4.608 -4.306 -3.955 -3.558 -3.120
-2.643 -2.132 -1.592 -1.029 -0.447 0.148 0.749 1.350
1.946 2.530 3.095 3.637 4.148 4.623 5.057 5.445 5.782
6.064 6.288 6.450 6.548 6.580 6.546 6.444 6.275 6.039
5.740 5.378 4.956 4.479 3.951 3.376 2.760 2.109 1.429
0.726 0.009 -0.717 -1.444 -2.164 -2.870 -3.555 -4.210
-4.830 -5.408 -5.936 -6.409 -6.822 -7.170 -7.448 -7.653
-7.783 -7.834 -7.806 -7.698 -7.511 -7.245]

Listing B.2: Sample output for Linear time series (correctly classified)

First, let's analyze the time series and make some observations:

1. The time series starts at a negative value, increases
↪ linearly, and ends at a positive value.
2. There is no apparent periodicity or oscillation in the time
↪ series.
3. There are no sudden jumps or changes in the time series,
↪ indicating a smooth function.

Now, let's go through the functions in the mapping and see which

↪ one matches our observations:

- 'gaussian_wave': This function has a periodic oscillation,
↪ which doesn't match our observations.
- 'exp': This function increases exponentially, which doesn't
↪ match the linear increase in our observations.
- 'linear_cos': This function has a periodic oscillation, which
↪ doesn't match our observations.
- 'linear': This function increases linearly, which matches our
↪ observations.
- 'sine': This function has a periodic oscillation, which doesn't
↪ match our observations.
- 'sinc': This function has a periodic oscillation, which doesn't
↪ match our observations.
- 'beat': This function has a periodic oscillation, which doesn't

- ↪ match our observations.
- 'sigmoid': This function has a sigmoid shape, which doesn't
 - ↪ match our observations.
- 'log': This function increases logarithmically, which doesn't
 - ↪ match the linear increase in our observations.
- 'x_times_sine': This function has a periodic oscillation, which
 - ↪ doesn't match our observations.
- 'square': This function has a parabolic shape, which doesn't
 - ↪ match our observations.

Based on our observations and analysis, the function that most
↪ likely generated the time series is 'linear'.

Answer: linear

Listing B.3: Sample output for Exp time series (correctly classified)

First, let's analyze the time series.

The time series starts with a value of 0.148 and ends with a

↪ value of 8.063. This indicates that the series is

↪ monotonically increasing.

Now, let's look at the differences between consecutive values.

↪ The differences are small and seem to be increasing as the

↪ series progresses. This suggests that the series is not

↪ only increasing but also accelerating.

With these observations, let's analyze the given functions:

1. gaussian_wave: This function represents a wave that is
↪ localized around a specific point. It does not match the
↪ characteristics of the time series.
2. exp: This function represents an exponential growth, which
↪ matches the increasing and accelerating characteristics of
↪ the time series.
3. linear_cos: This function represents a linear function with an
↪ added cosine term. It does not match the accelerating
↪ characteristic of the time series.
4. linear: This function represents a linear growth, which does
↪ not match the accelerating characteristic of the time
↪ series.

5. `sine`: This function represents a sinusoidal wave, which does
↪ not match the characteristics of the time series.
6. `sinc`: This function represents a sinc function, which does not
↪ match the characteristics of the time series.
7. `beat`: This function represents a beat pattern, which does not
↪ match the characteristics of the time series.
8. `sigmoid`: This function represents a sigmoid curve, which does
↪ not match the characteristics of the time series.
9. `log`: This function represents a logarithmic growth, which does
↪ not match the accelerating characteristic of the time
↪ series.
10. `x_times_sine`: This function represents a product of a linear
↪ function and a sinusoidal function, which does not match
↪ the characteristics of the time series.
11. `square`: This function represents a quadratic growth, which
↪ does not match the characteristics of the time series.

Based on our analysis, the `exp` function is the most likely
↪ candidate to have generated the time series.

Answer: `exp`

Listing B.4: Sample output for Square time series (correctly classified)

Observations about the time series:

1. The values are positive and continuous.
2. The values start from a high value and gradually decrease
↪ until they reach a minimum value near the middle of the
↪ series.
3. After the minimum value, the values gradually increase again
↪ until they reach a high value near the end of the series.
4. The overall trend of the time series seems to be symmetric.

Now, let's analyze the given functions and their characteristics:

1. gaussian_wave: Symmetric, bell-shaped curve, values are
↪ positive and continuous.
2. exp: Exponential growth, not symmetric.
3. linear_cos: Linear trend with oscillations, not symmetric.
4. linear: Linear growth, not symmetric.
5. sine: Oscillating function, not symmetric.
6. sinc: Symmetric, sinc function has a peak at the center and
↪ oscillates on both sides.
7. beat: Combination of two sine waves with different
↪ frequencies, not symmetric.
8. sigmoid: S-shaped curve, not symmetric.
9. log: Logarithmic growth, not symmetric.
10. x_times_sine: Product of a linear function and sine function,
↪ not symmetric.

11. square: Quadratic function , symmetric , parabolic shape .

Based on our observations , the most likely function to have

↪ generated this time series is the 'square' function . It is

↪ symmetric , has a parabolic shape , and the values are

↪ positive and continuous .

Answer: square

Listing B.5: Sample output for Gaussian_wave time series (misclassified)

We can start by analyzing the time series and making observations

↪ about its characteristics:

1. The time series starts and ends at 0, with values close to 0
↪ in the beginning and end.
2. The time series has a single peak and a single valley,
↪ reaching a maximum value around 0.85 and a minimum value
↪ around -1.
3. The time series is not periodic, as it doesn't repeat its
↪ pattern.

Now, let's go through the functions one by one and see which one

↪ matches these observations:

- 'gaussian_wave': This function is a product of a Gaussian and a
↪ sinusoidal function, which could give us a single peak and
↪ valley. However, it is periodic, so it doesn't match
↪ observation 3.
- 'exp': This function is an exponential function, which doesn't
↪ match the observations as it doesn't have a peak and valley.
- 'linear_cos': This function is a linear function combined with
↪ a cosine function, which could give a single peak and
↪ valley. However, it is also periodic, so it doesn't match
↪ observation 3.
- 'linear': This function is a linear function, which doesn't

- ↪ have a peak and valley , so it doesn't match observation 2.
- 'sine ': This function is a sinusoidal function , which is
 - ↪ periodic and doesn't match observation 3.
- 'sinc ': This function is a sinc function , which has multiple
 - ↪ peaks and valleys , so it doesn't match observation 2.
- 'beat ': This function is a product of two sinusoidal functions ,
 - ↪ which could give a single peak and valley . However, it is
 - ↪ also periodic , so it doesn't match observation 3.
- 'sigmoid ': This function is a sigmoid function , which matches
 - ↪ all the observations . It starts and ends at 0, has a single
 - ↪ peak and valley , and is not periodic .
- 'log ': This function is a logarithmic function , which doesn't
 - ↪ have a peak and valley , so it doesn't match observation 2.
- 'x_times_sine ': This function is a product of a linear function
 - ↪ and a sinusoidal function , which could give a single peak
 - ↪ and valley . However, it is also periodic , so it doesn't
 - ↪ match observation 3.
- 'square ': This function is a quadratic function , which doesn't
 - ↪ have a peak and valley , so it doesn't match observation 2.

Based on these observations , the most likely function to have

↪ generated the time series is 'sigmoid'.

Answer: sigmoid

Listing B.6: Sample output for Linear_Cos time series (misclassified)

Let's analyze the time series step by step:

1. The time series starts at a negative value, increases to a
↪ positive value, and then decreases back to a negative
↪ value. This suggests that the function might have a
↪ symmetric or periodic behavior.
2. The time series has a peak in the middle, which indicates that
↪ the function might have a single maximum or minimum point.
3. The time series does not show an exponential growth or decay,
↪ which rules out 'exp', 'log', and 'sigmoid' functions.
4. The time series does not have a linear pattern, which rules
↪ out the 'linear' function.
5. The time series does not show a clear periodic pattern, which
↪ rules out 'sine', 'linear_cos', and 'beat' functions.
6. The time series does not show a sinc pattern, which rules out
↪ the 'sinc' function.
7. The time series does not show a clear x_times_sine pattern,
↪ which rules out the 'x_times_sine' function.

8. The time series does not show a clear parabolic pattern, which
↪ rules out the 'square' function.

Based on the above observations, the only remaining function is
↪ 'gaussian_wave'. Therefore, the most likely function to
↪ have generated the time series is:

Answer: gaussian_wave

C | APPENDIX: LANGUAGE MODELING FOR 3D CRYSTAL DATA

C.1 TRAINING DETAILS

C.1.1 NUMERICAL FORMATTING

Notably, our approach to tokenization is distinctly different from prior work on modeling atomic structures with language models. Instead of using a special vocabulary and training models from scratch, we use LLaMA-2’s existing tokenizer. This choice allows us to easily process both encoded crystals and text data. In early experiments, we tried out many other approaches, including fine-tuning LLaMA-2 models with additional tokens specific to crystal data. These methods were more challenging to train and didn’t lead to any improvements over using a shared tokenizer. We include a set of example training losses below:

	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
Special Crystal Tokens	0.783	0.693	0.623	0.611	0.588
Shared Tokenization	0.457	0.432	0.424	0.401	0.385

There are many important decisions involved both in text formatting (e.g the choice of fractional or absolute coordinates) and augmentation of the input data (e.g. translation or permutation augmentations on coordinates). As a simple example, we provide average

validity numbers (using low temperature sampling) from earlier experiments on LLaMA-2 7B models trained with different formatting styles

Setting	Structural Validity	Compositional Validity
Fractional coords	91.4%	83.2%
Absolute coords	90.8%	80.5%
No permutations	92.5%	82.9%
With permutations	89.2%	81.7%

C.1.2 TRAINING WITH STOCHASTIC PROMPTS

In order to enable multi-task use of the fine-tuned LLMs, we train on a stochastically generated prompt. Two thirds of the time we provide the model with a generation task, in which the prompt consists of a basic instruction to generate a bulk material as a lattice and atom positions. We randomly sample a set of properties from the available descriptors of a given crystal and add any chosen ones (if any) to the prompt, using a small amount of wrapper text. The remaining one third of the time, we provide use the sampled crystal to construct and infilling task. We choose on element randomly from the set of elements in the composition and we construct a prompt that contain the string encoding of the crystal with this element replaced with [MASK]. The model then generates the replaced element as text following the prompt.

C.1.3 EXTENDED MATERIALS PROJECT DATASET

To facilitate text-conditional generation, we extend the original CDVAE training dataset with materials from Materials Project [118] as of April 2023. We filter out crystal with more than 30 atoms in the unit cell, which slow down training with minimal benefit to model performance, leaving a training set that contains 127609 crystal structures. The original

validation and test splits are left unchanged and all test/validation points are removed from the new training set.

C.1.4 TRAINING HYPERPARAMETERS AND DETAILS

We provide the training details per model:

- **LLaMA-2 7B**: Batch size of 256 for 65 epochs with a cosine annealed learning rate of 0.0005. LoRA rank 8 and alpha 32.
- **LLaMA-2 13B**: Batch size of 256 for 44 epochs with a cosine annealed learning rate of 0.0005. LoRA rank 8 and alpha 32.
- **LLaMA-2 70B**: Batch size of 32 for 21 epochs with a cosine annealed learning rate of 0.0005. LoRA rank 8 and alpha 32.

Limitations around available compute lead to our use of differing batch sizes and total number of epochs for each model. Ideally, we would train all models with the largest batch sized used among all models and would train all models for the same number of epochs (the maximum used by any model). At the same time, we wanted to properly demonstrate the full potential of all model sizes and therefore chose to present results for the best model we were able to train at each model size.

C.1.5 ROLE OF TEXT PRETRAINING

Text pretraining is essential to our method for two reasons.

1. It would be impractically expensive or computationally infeasible to train models with up to 70B parameters from scratch on our data. Using a pretrained model with LoRA [112] offers the benefits of model scale while maintaining tractability and limiting overfitting, as the actual number of trainable parameters can be relatively small.

2. Pretraining on text data yields a model that can be conditioned on text for free, and text conditioning opens up a huge new realm of exciting possibilities, like conditioning samples on desired properties. It would be challenging to achieve a similar result from scratch without significantly expanding the size of the dataset (to improve general text understanding) and without essentially training a general-purpose language model in the process.

To better understand the first point, let’s quickly review the exact details of the finetuning procedure. We are using low-rank adapters (LoRA), as opposed to end-to-end finetuning, and this means we are adding a small number of additional parameters to an existing, frozen model. The easiest way to see the difference between this approach and training a model from scratch—as in [72]—is to compare the training loss over the first few epochs of training.

Model	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
GPT-2 (from scratch)	0.946	0.878	0.807	0.757	0.740
LLaMA-13B (LoRA)	0.457	0.432	0.424	0.401	0.385
LLaMA-70B (LoRA)	0.402	0.344	0.325	0.305	0.296

If we attempt to run LoRA finetuning with randomly initialized parameters for the LLaMA-2 7B model we observe an immediate and significant difference in the training losses:

Model	1 Iter	0.33 Epochs	0.66 Epochs	1 Epoch
Random	13.46	1.53	0.81	0.78
Pre-trained	1.57	0.47	0.41	0.39

While LoRA finetuning is tractable because 99.95% of the model is frozen, finetuning a LLaMA-2 model end-to-end in half-precision would require at least 4 times as many GPUs, making it infeasible for all but a handful of researchers. When using LoRA, even though the base models are large the number of trainable parameters is very small. In fact, the LLaMA-2 7B model has less trainable parameters than one of the baseline methods we compared

(CDVAE) [232]. The number of trainable parameters for each of our models and the baseline models is shown below:

Model	Trainable parameters (millions)	Percentage of total
CDVAE	4.5	100%
LM-CH/AC	1-100	100%
LLaMA-2 7B	3.5	0.05%
LLaMA-2 13B	6.5	0.05%
LLaMA-2 70B	35	0.05%

C.2 MODEL EVALUATION

C.2.1 EVALUATION WITH ML POTENTIALS AND DFT

Approximating E_{hull} from the energies of known materials in Materials Project requires a consistent correction scheme. We touch on some of the details here.

M3GNET Importantly, M3GNet was trained on the total energy of VASP calculations in the Materials Project dataset, so the results were expected to be consistent with the correction schemes and absolute energies in the experiments section.

VASP To be consistent with the Materials Project settings (e.g. the PBE functional, DFT/DFT+U as appropriate, consistent pseudopotentials, etc). We did a single relaxation for every candidate structure using the default parameters in MPRelaxSet [167]. VASP relaxations were run using the GPU-accelerated VASP6 code.

In both situations, the total energies were corrected using the MP2020 compatibility scheme, which was important to maintain consistency when calculating formation energies, and allow the use of varying functionals (DFT/DFT+U) for different materials.

C.2.2 STABILITY CHECKS AND PERCENTAGES

To calculate the percentage of metastable compounds, we take all samples and remove samples that are invalid under the basic structure and composition checks. We then run relaxations with M3GNet and obtain the final relaxation energies. The final percentage takes into account both the rate of validity (used to perform the initial filtering), and the rate of compounds with $\hat{E}_{\text{hull}} < 0.1$, as determined by the convex hull calculation using the M3GNet relaxation energy. To calculate the VASP percentage, we select materials determined to be metastable M3GNet and run VASP with default setting. We then report the percentage of the materials with $\hat{E}_{\text{hull}} < 0.0$.

C.2.3 TRADE-OFFS IN SAMPLING

We note that modulating stability with sampling parameters like temperature and nucleus size has a significant effect on the coverage properties of the resulting samples. We illustrate the trade-offs between stability and coverage in Figure C.1. Coverage most likely decreases because nucleus size and temperature collapse the distribution around samples with high likelihood, which are also more likely to be valid or stable. Notably, LLaMA-2 70B appears to demonstrate the best trade-offs, possibly indicating a likelihood model that corresponds better to both the underlying properties of stability and the full, diverse distribution of structures.

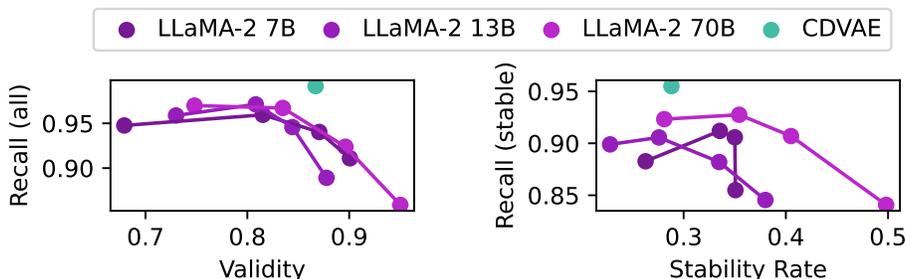


Figure C.1: Validity and rate of stability depend on sampling hyper-parameters. Lowering the temperature or restricting the nucleus size leads to significant improvements in validity/stability but incurs a cost to coverage of a held-out test set (recall). Fine-tuned LLaMA-2 70B displays the best trade-off between coverage and stability, generating materials that are both stable and diverse.

C.2.4 “HALLUCINATION” EXAMPLES

LLaMA-2 7B:

```
# generated using pymatgen
data_Met8 (Cu2N)5
__symmetry_space_group_name_H-M 'P 1'
__cell_length_a 5.0000
__cell_length_b 5.0000
__cell_length_c 5.0000
__cell_angle_alpha 90.0000
__cell_angle_beta 90.0000
__cell_angle_gamma 90.0000
__symmetry_Int_Tables_number 1
__chemical_formula_structural Met8(Cu2N)5
__chemical_formula_sum 'Met8 Cu10 N5'
__cell_volume 125.0000
__cell_formula_units_Z 1
loop_
__symmetry_equiv_pos_site_id
__symmetry_equiv_pos_as_xyz
1 'x, y, z'
loop_
__atom_site_type_symbol
__atom_site_label
__atom_site_symmetry_multiplicity
__atom_site_fract_x
__atom_site_fract_y
__atom_site_fract_z
__atom_site_occupancy
Cu Cu0 1 1.8300 0.3900 1.0000 1
Cu Cu1 1 0.8300 0.4900 1.0000 1
Cu Cu2 1 0.8300 0.9900 0.5000 1
Cu Cu3 1 0.6300 0.1900 0.2000 1
Cu Cu4 1 0.2300 0.7900 0.2000 1
Cu Cu5 1 0.6300 0.7000 0.3100 1
Cu Cu6 1 0.2300 0.1900 0.3000 1
Cu Cu7 1 1.0000 0.8900 0.7000 1
Cu Cu8 1 1.0000 0.3900 0.2000 1
Cu Cu9 1 0.4900 0.8900 0.7000 1
Met0+ Met10 1 0.6300 0.6000 1.0000 1
Met0+ Met11 1 0.4000 0.4700 0.4700 1
Met0+ Met12 1 0.4000 1.0000 0.9800 1
Met0+ Met13 1 1.0000 0.2200 0.9700 1
Met0+ Met14 1 1.0000 0.6300 0.5000 1
Met0+ Met15 1 0.2300 0.2200 0.6000 1
Met0+ Met16 1 1.0000 0.0000 0.6100 1
Met0+ Met17 1 0.6300 0.1000 0.5000 1
N N18 1 0.1200 0.7000 0.8000 1
N N19 1 0.2300 0.5900 0.2000 1
N N20 1 0.2300 0.1900 0.7000 1
N N21 1 0.4900 0.2100 0.1000 1
N N22 1 0.4800 0.6100 0.6000 1
```

```
data_L3Li
__symmetry_space_group_name_H-M 'P 1'
__cell_length_a 5.1000
__cell_length_b 7.1000
__cell_length_c 7.4000
__cell_angle_alpha 84.0000
__cell_angle_beta 68.0000
__cell_angle_gamma 68.0000
__symmetry_Int_Tables_number 1
__chemical_formula_structural L3Li
__chemical_formula_sum 'Li2 Li4'
__cell_volume 230.15214369
__cell_formula_units_Z 4
loop_
__symmetry_equiv_pos_site_id
__symmetry_equiv_pos_as_xyz
1 'x, y, z'
loop_
__atom_site_type_symbol
__atom_site_label
__atom_site_symmetry_multiplicity
__atom_site_fract_x
__atom_site_fract_y
__atom_site_fract_z
__atom_site_occupancy
Li Li0 1 0.7100 0.4000 0.8300 1
Li Li1 1 0.2200 0.3700 0.3600 1
Li Li2 1 0.7100 0.8900 0.3300 1
Li Li3 1 0.2100 0.8700 0.8600 1
L0+ L4 1 1.0000 0.6300 0.6900 1
L0+ L5 1 0.5100 0.1400 0.6600 1
L0+ L6 1 0.9600 0.5700 0.1700 1
L0+ L7 1 0.4700 0.0700 0.1700 1
L0+ L8 1 0.9800 0.6100 0.1400 1
L0+ L9 1 0.4900 0.1100 0.1400 1
L0+ L10 1 1.0000 0.1000 0.6800 1
L0+ L11 1 1.0000 0.1100 0.1500 1
L0+ L12 1 0.4700 0.5500 0.1800 1
L0+ L13 1 1.0000 0.5800 0.6800 1
L0+ L14 1 0.4700 0.0600 0.6700 1
L0+ L15 1 1.0000 0.1300 0.1700 1
```

LLaMA-2 13B:

```
data_LeB7(NO3)2
_symmetry_space_group_name_H-M 'P 1'
_cell_length_a 5.9000
_cell_length_b 5.9000
_cell_length_c 5.9000
_cell_angle_alpha 59.0000
_cell_angle_beta 59.0000
_cell_angle_gamma 59.0000
_symmetry_Int_Tables_number 1
_chemical_formula_structural LeB7(NO3)2
_chemical_formula_sum 'Le1 B7 N2 O6'
_cell_volume 141.91223582
_cell_formula_units_Z 1
loop_
_symmetry_equiv_pos_site_id
_symmetry_equiv_pos_as_xyz
1 'x, y, z'
loop_
_atom_site_type_symbol
_atom_site_label
_atom_site_symmetry_multiplicity
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
_atom_site_occupancy
Le0+ Le0 1 0.7100 0.5000 0.1700 1
B B1 1 0.3800 0.1600 0.0200 1
B B2 1 0.4600 0.1600 0.5700 1
B B3 1 0.4600 0.7200 0.5700 1
B B4 1 0.0400 0.7900 0.6500 1
B B5 1 1.0000 0.2500 0.6500 1
B B6 1 0.0000 0.7900 0.0900 1
B B7 1 0.0000 0.1600 0.6500 1
N N8 1 0.6200 0.5700 0.9800 1
N N9 1 0.0600 0.3300 0.2500 1
O O10 1 0.5500 0.7600 0.7100 1
O O11 1 0.1800 0.5400 0.6100 1
O O12 1 0.4300 0.9500 0.5400 1
O O13 1 0.9400 0.1100 0.9600 1
O O14 1 0.6400 0.7700 0.2900 1
O O15 1 0.3000 0.3800 0.1300 1
```

```
data_MandeGd2O4
_symmetry_space_group_name_H-M 'P 1'
_cell_length_a 3.6000
_cell_length_b 3.6000
_cell_length_c 5.9000
_cell_angle_alpha 90.0000
_cell_angle_beta 90.0000
_cell_angle_gamma 90.0000
_symmetry_Int_Tables_number 1
_chemical_formula_structural MandeGd2O4
_chemical_formula_sum 'Mande1 Gd2 O4'
_cell_volume 76.46400000
_cell_formula_units_Z 1
loop_
_symmetry_equiv_pos_site_id
_symmetry_equiv_pos_as_xyz
1 'x, y, z'
loop_
_atom_site_type_symbol
_atom_site_label
_atom_site_symmetry_multiplicity
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
_atom_site_occupancy
Gd Gd0 1 0.8200 0.2300 0.1500 1
Gd Gd1 1 0.8200 0.2300 0.6300 1
Mande0+ Mande2 1 0.3200 0.7300 0.8900 1
O O3 1 0.8200 0.7300 0.4100 1
O O4 1 0.3200 0.7300 0.1000 1
O O5 1 0.3200 0.2300 0.3900 1
O O6 1 0.8200 0.7300 0.7900 1
```

LLaMA-2 70B:

```

data_Ln3BO4
__symmetry_space_group_name_H-M 'P 1'
__cell_length_a 5.3000
__cell_length_b 5.9000
__cell_length_c 5.3000
__cell_angle_alpha 62.0000
__cell_angle_beta 90.0000
__cell_angle_gamma 90.0000
__symmetry_Int_Tables_number 1
__chemical_formula_structural Ln3BO4
__chemical_formula_sum 'Ln3 B1 O4'
__cell_volume 146.33178751
__cell_formula_units_Z 1
loop_
  __symmetry_equiv_pos_site_id
  __symmetry_equiv_pos_as_xyz
  1 'x, y, z'
loop_
  __atom_site_type_symbol
  __atom_site_label
  __atom_site_symmetry_multiplicity
  __atom_site_fract_x
  __atom_site_fract_y
  __atom_site_fract_z
  __atom_site_occupancy
Ln0+ Ln0 1 0.1800 0.0600 0.9900 1
Ln0+ Ln1 1 0.6800 0.5600 0.9900 1
Ln0+ Ln2 1 0.1800 0.5600 0.4900 1
B B3 1 0.6800 0.0600 0.4900 1
O O4 1 0.6800 0.3300 0.1500 1
O O5 1 0.1800 0.2800 0.1800 1
O O6 1 0.6800 0.7800 0.8000 1
O O7 1 0.1800 0.8300 0.8500 1

```

```

data_Gro15Nd4
__symmetry_space_group_name_H-M 'P 1'
__cell_length_a 7.0000
__cell_length_b 7.0000
__cell_length_c 6.9000
__cell_angle_alpha 71.0000
__cell_angle_beta 71.0000
__cell_angle_gamma 69.0000
__symmetry_Int_Tables_number 1
__chemical_formula_structural Gro15Nd4
__chemical_formula_sum 'Gro15 Nd4'
__cell_volume 289.96945358
__cell_formula_units_Z 1
loop_
  __symmetry_equiv_pos_site_id
  __symmetry_equiv_pos_as_xyz
  1 'x, y, z'
loop_
  __atom_site_type_symbol
  __atom_site_label
  __atom_site_symmetry_multiplicity
  __atom_site_fract_x
  __atom_site_fract_y
  __atom_site_fract_z
  __atom_site_occupancy
Nd Nd0 1 0.5600 0.5700 0.7800 1
Nd Nd1 1 0.7500 0.7500 0.5600 1
Nd Nd2 1 0.1700 0.1700 0.1400 1
Nd Nd3 1 0.9500 0.9500 0.3800 1
Gro0+ Gro4 1 0.7600 0.2300 0.3000 1
Gro0+ Gro5 1 0.1200 0.4800 1.0000 1
Gro0+ Gro6 1 0.3800 0.8700 0.1000 1
Gro0+ Gro7 1 0.0300 0.6600 0.8400 1
Gro0+ Gro8 1 0.6500 0.1700 0.6400 1
Gro0+ Gro9 1 0.5600 0.0600 0.7400 1
Gro0+ Gro10 1 0.9200 0.5000 0.1600 1
Gro0+ Gro11 1 0.4900 0.7400 0.2200 1
Gro0+ Gro12 1 0.2400 0.1000 0.5800 1
Gro0+ Gro13 1 0.9100 0.2700 0.6200 1
Gro0+ Gro14 1 0.4000 0.6100 0.4600 1
Gro0+ Gro15 1 0.2900 0.2900 0.4200 1
Gro0+ Gro16 1 0.4500 0.9200 0.9400 1
Gro0+ Gro17 1 0.9900 0.1300 0.0200 1
Gro0+ Gro18 1 0.8400 0.5100 0.8200 1

```

C.2.5 INCREASE IN PERPLEXITY UNDER TRANSFORMATION (IPT)

Although there are existing metrics for invariance and equivariance in neural networks, language models pose unique challenges because of their discrete tokens, which do not change smoothly under continuous transformations. Though it might be possible to compute a meaningful analogue of the Lie derivative [90], or similar metrics, through interpolation of word embeddings, we decide to adopt a simpler metric (IPT), which still highlights significant differences between base models. We calculate IPT for each model using 500 test datapoints and 20 randomly translation sampled as fraction coordinates from a uniform distribution per dimension. The translations themselves are implemented in PyMatgen and respect periodic boundary conditions [167]. In order to combine the IPT values in a meaningful way across different datapoints, we normalize their values by the mean perplexity over transformations. Thus datapoints which happen to have large perplexity, and therefore naturally large potential changes in perplexity, do not drown out points with small perplexity.

C.2.6 DIVERSITY AND NOVELTY CALCULATION

Following [232], we calculate diversity as the pairwise distance between samples using a featurization of structure and composition. To calculate novelty, we also featurize the training dataset and calculate the distance to the nearest element of the training set for each sample. A sample is considered novel if the nearest element in the training set is above a threshold. We use a structural distance cutoff of 0.1 and composition distance cutoff of 2. In addition to novelty of structure and composition individual, we also consider the overall novelty of a crystal, where overall novelty is determined by having either a new structure or a new composition. All metrics are calculated on filtered samples that M3GNet qualifies as metastable. We report metrics on metastable samples because these numbers are more practically relevant and because the samples are more likely to contribute meaningful

variation, instead of being different from the training set and each other simply because they are wildly invalid. We normalize all diversity and novelty values by corresponding value for the test set to provide a sense for the underlying data distribution.

C.2.7 SAMPLING SPEED

Although LLMs might seem like computational overkill at face value, batching for large-scale sampling allows LLaMA models to have comparable computational overhead to competing approaches. Making exact comparisons between LLaMA models and CDVAE are slightly challenging because of available hardware and differences in compatibility. We ran experiments primarily on A100 GPUs, while the publicly available code for CDVAE cannot be run on an A100 and reports results on a RTX2080 Ti.

We provide two analyses for the sampling rate of LLaMA models, one from experiments we ran on a single A100 and alternative using third-party numbers for LLaMA models deployed on AWS instances.

LOCAL ANALYSIS We obtain benchmark LLaMA-2 sampling times by running 5 batched generations and computing the average time to completion. We then use these numbers to calculate the equivalent time to sample 10,000 structures. In practice, we used distributed sampling on a cluster, so reporting our direct times to compute 10,000 samples would be less informative. We use the maximum batch size that we can fit on an A100 GPU with each model without causing out-of-memory (OOM) errors during sampling. The batch sizes were {7B: 512, 13B: 256, 70B: 128}. To compare CDVAE with our results we perform a rough, but generous, conversion of their results to an A100 GPU. We multiply their rate of sampling by 16, to account for the 2x faster rate of operations [16] and approximately 8 times larger GPU memory (allowing for large batch sizes and utilization rates). We report the intermediate numbers and calculations below. The final rates for metastable samples are

shown in Figure 5.5.

Model	Batch size	Seconds / batch	Samples / hour	Hours / 10,000 crystals
CDVAE	512	n/a	n/a	1.260
LLaMA-2 7B	512	27.18	67814	0.147
LLaMA-2 13B	256	38.24	24100	0.414
LLaMA-2 70B	128	52.52	8774	1.139

AWS ANALYSIS Considering AWS as the deployment environment, we can build on a recent benchmark on a cloud instance with 8 A100 GPUs (ml.p4d.12xlarge) [188], which found that LLaMA-2 13B achieved 0.416 hr/1M tokens and LLaMA-2 70B achieved 0.864 hr/1M tokens. One crystal is around 100 tokens on average, so the throughput for 10,000 crystals is the same as for 1M tokens. For comparison, we use CDVAE and its recorded runtimes for generating 10,000 crystals on a single RTX2080 Ti GPU [232]. To obtain the final numbers, we adjust for the number of GPUs (8) and a 2x improvement from RTX2080 Ti to A100 GPUs [16].

Model	Hours / 10,000 crystals	Hours / 10,000 metastable (M3GNet) crystals
CDVAE	0.363	1.260
LLaMA-2 13B	0.416	1.094
LLaMA-2 70B	0.864	1.728

We see that LLaMA-2 13B actually has a comparable computational overhead to prior work, and LLaMA-2 70B is only slightly higher. When considering the rate of stable materials generated by each method, we see that LLaMA-2 13B actually has a higher throughput than CDVAE.

C.3 TEMPLATE METHOD BASELINE

We provide code in Listing 1 implementing construction of the physically-inspired element swap table. This table is used by both the template method and the LLM-guided

sampling method to constrain search to elements that are physically plausible. Listing 2 shows our implementation of a basic template method with uniform sampling. The LLM-guided procedure is mostly identical, except with uniform sampling of the swap element changed for sampling from a distribution obtained from the LLM with an infilling prompt (and modulated with temperature parameter τ)

Listing C.1: Self contained code to construct the template method table which can be used to proposed mutations for local optimization around an existing material. The same table can be used in tandem with a language model to provide sampling constraints (i.e. eliminate elements which are very physically unlikely).

```
import os

import random

import pandas as pd

import numpy as np

from pymatgen.core import Element

from pymatgen.core.structure import Structure

from m3gnet.models import Relaxer

def find_similar_elements(target_element, elements,
    ↪ tolerance=0.1):
    similar_elements = []
    for state, radius in target_element.ionic_radii.items():
        for el in elements:
            if state in el.ionic_radii:
                radius_diff = abs(radius - el.ionic_radii[state])
                if radius_diff < tolerance and el.symbol !=
                    ↪ target_element.symbol:
```

```

        similar_elements.append((el.symbol, state,
                                ↪ radius_diff))
    return sorted(similar_elements, key=lambda x: x[2])

def make_swap_table():
    elements = [Element(el) for el in Element]

    swap_table = {}

    for el in elements:
        swap_table[el.symbol] = [
            x[0] for x in find_similar_elements(el, elements)
        ]

    return swap_table

```

Listing C.2: Self contained code implementing a template method with uniform sampling. Our language model procedure is essentially the same but replaces uniform sampling with logits from a prompted language model. This language model can use the context from the rest of the crystal structure to propose a mutation instead of choosing a mutation completely at random.

```

def propose_new_structures(cif_str, swap_table, max_swaps=1):
    struct = Structure.from_str(cif_str, fmt="cif")

    elements = [el.symbol for el in struct.species]
    swappable_elements = [
        el for el in elements if el in swap_table and
        ↪ len(swap_table[el]) > 0
    ]

```

```

num_possible_swaps = sum([len(swap_table[el]) for el in
    ↪ swappable_elements])
num_swaps = min(num_possible_swaps, max_swaps)

relaxer = Relaxer()
new_bulks = []
for _ in range(num_swaps):
    old_el = random.choice(swappable_elements)
    possible_new = swap_table[old_el]
    new_el = random.choice(possible_new)

    new_bulk = struct.copy()
    new_bulk.replace_species({old_el: new_el})

    relax_results = relaxer.relax(new_bulk)
    final_structure = relax_results['final_structure']
    final_relaxed_energy =
    ↪ relax_results['trajectory'].energies[-1]

    new_bulks.append(dict(
        cif=final_structure.to(fmt="cif"),
        energy=final_relaxed_energy
    ))

new_bulks = pd.DataFrame(new_bulks)

```

```
return new_bulks
```

D | APPENDIX: CHALLENGES IN APPLYING LANGUAGE MODELS TO NUMERICAL DATA

D.1 CONDITIONAL VS UNCONDITIONAL MODELING

In standard language modeling and in the supervised finetuning of language models, the joint distribution of the data is modeled, enforced by minimizing the NLL $-\log p(x) = \sum_i -\log p(x_i|x_{<i})$. For many of the problems we consider on structured numerical data, there is an explicit input-output structure, and we are only interested in the conditional distribution $p(y|x)$ for e.g. numerical outputs computed from a point cloud. Posed as sequence modeling we could also state $p(y|x)$ as $p(x_{>i}|x_{\leq i})$. While learning the joint distribution also implies learning the conditional distribution in the abstract, high complexity and variance in $p(x)$ can mean that signal in y gets drowned out in the unnecessary task of modeling x . In many cases, the entropy of the output $H(y|x)$ is much lower than $H(x)$, and thus the model prioritizes x . For example, learning the distribution of all rotations of a molecule might be much more complicated than just learning to distinguish high

Table D.1: MAE (\downarrow) values for training with and without masking, both from scratch and finetuning.

Type	w	w/out
Scratch	0.168	0.154
Finetune	0.456	0.508

and low energy configurations. When learning jointly on $p(x_{>i}|x_{\leq i})$ and $p(x_{\leq i})$, the gradient signals of each term compete, leading to slower learning of $p(x_{>i}|x_{\leq i})$ than in models that are explicitly conditional. Even if $p(x_{\leq i})$ is modeled perfectly, the random variation in $x_{\leq i}$ introduces unnecessary noise in the gradients, which slows down learning as we show in Appendix D.2.

To test this effect, we train models, both from scratch and fine-tuning text pretrained models, on the energy (from coordinates) task with loss masking to optimize only $p(x_{>i}|x_{\leq i})$. The results are displayed in Table D.1. While masking helps fine-tuning which has relatively few gradient steps (1 epoch), masking does not help when training from scratch (100 epochs).

D.2 LEARNING SPEEDUP FROM LOSS MASKING

When learning $p(y|x)$, the training convergence can be substantially slowed down when including the $p(x)$ loss contribution.

Consider the loss for a single data point with a random label:

$$L = -y^\top \log \sigma(f_\theta(x))$$

where $f(x)$ is the mapping to the log softmax of the logits of the model, σ is the softmax function, and y is the one-hot random label vector (among the V classes).

The gradient is

$$\nabla_\theta L = y^\top [I - \mathbb{K}\sigma^\top]J$$

where J is the Jacobian of the network outputs with respect to θ . $\mathbb{E}[y] = \mathbb{K}/V$ giving an expectation of

$$\mathbb{E}[\nabla_\theta L] = (1/V)\mathbb{K}^\top [I - \mathbb{K}\sigma^\top]J.$$

The gradient is 0 when the model predicts a uniform distribution $\sigma = \mathbb{K}/V$, and we will

consider perturbations around this point.

From $\mathbb{E}[yy^\top] = I/V$ covariance is given by

$$\mathbb{E}[\nabla L \nabla L^\top] = (1/V)J^\top [I - \mathbb{K}\sigma^\top]^\top [I - \mathbb{K}\sigma^\top]J.$$

Letting $\sigma = \mathbb{K}/V$, the gradient norm is

$$\mathbb{E}[\|\nabla L\|^2] = (1/V)\text{Tr}(PJJ^\top),$$

for $P = [I - \mathbb{K}\mathbb{K}^\top/V]$.

The convergence of SGD on convex problems can be written in terms of the expectation of the norm of the gradient. Over T timesteps with learning rate η and batch size B , the convergence can be written (see e.g. Shalev-Shwartz and Ben-David [190]) as

$$\frac{1}{T} \sum_{t=1}^T \mathbb{E}[\|\nabla L(\theta_t)\|^2] \leq 2 \frac{L(\theta_0) - L(\theta_*)}{\eta T} + (\eta\sigma^2/B), \quad (\text{D.1})$$

where $\sigma^2 = \sup_{\theta} \mathbb{E}[\|\nabla L(\theta, y) - \mathbb{E}[\nabla L(\theta, y)]\|^2]$ with expectations taken over the distribution of y . The convergence of SGD is limited by this noisy ball term $(\eta\sigma^2/B)$, and for a fixed learning rate cannot improve upon that limit as $T \rightarrow \infty$.

If $\mathbb{E}[\|\nabla L(\theta_*, y) - \mathbb{E}[\nabla L(\theta_*, y)]\|^2] = \mathbb{E}[\|\nabla L(\theta_*, y)\|^2] = (1/V)\text{Tr}(PJJ^\top)$, then $\sigma^2 \geq (1/V)\text{Tr}(PJJ^\top)$, therefore increasing the size of the noisy ball and loss value that SGD converges to.

For the $p(y, X)$ vs $p(y|X)$ scenario, $p(y, X)$ contains the additional random content of X even if y is a deterministic function of X . This random content when mixed in to the negative log likelihood objective increases the size of the noisy ball, slowing down convergence.

D.3 GNN TRAINING DETAILS

We use a batch size of 96 and a learning rate of 0.001 for 200 epochs on the HOMO prediction task and for 50 epochs on the synthetic energy prediction task from coordinates only. We use a learning rate of 0.0005 for 100 epochs on the energy prediction task from distances. In all tasks, we use weight decay of 10^{-16} and a cosine decay on the learning rate. We do not use any normalization on the target function, and we add in an additional tanh activation function for stability.

D.4 xVAL ABLATIONS

In an attempt to understand the dominance of the continuous approach, we perform two additional ablations on the input and output of xVal by replacing them with their discrete counterpart, as shown in Figure D.1. **Continuous Input** ablates the benefit of passing numbers directly into the model, without needing to parse inputs from a sequence of tokens, while **Continuous Output** ablates the benefit of using a continuous loss function, while still using discrete inputs. Figure 6.6 (left) shows the result on the hardest numerical tasks, which indicates that neither design choice explains the strong performance of xVal in isolation, though continuous variants still outperform discrete approaches on 3D structure tasks.

D.5 SCALING EXPERIMENT

We train models with the following hyperparameter values:

To compute FLOPs we use the following formula:

$$\text{FLOPs} = 6 * (P * 10^6) * (256 * 1000 * S)$$

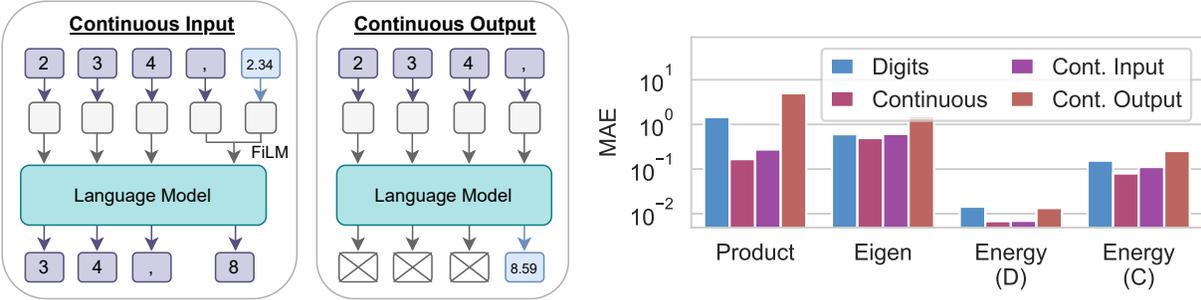


Figure D.1: (Left) We include ablations on xVal to explore the effect of working with discrete versus continuous inputs and the corresponding loss functions. (Right) To understand the performance of xVal, we perform an ablation the output and input with discrete tokens to understand if continuous inputs or continuous outputs (continuous loss) is the origin of improved performance. Both ablations hurt performance, but continuous inputs appear to be more helpful than continuous outputs.

Approx. # parameters (millions)	Hidden size	Intermediate size	Attention heads	Hidden layers
7	128	512	4	3
14	224	896	7	4
20	288	1152	7	5
38	448	1792	7	6
52	512	2048	8	8
66	576	2304	9	9

Table D.2: Hyperparameter values for model architecture for from-scratch language model training runs.

where P is the parameter count in millions and S is the number of gradient steps.

D.6 HYPERPARAMETER SETTINGS

D.6.1 FROM-SCRATCH MODELS

For our language model training from-scratch, we use the following hyperparameter values:

D.6.2 FINE-TUNED MODELS

For our language model fine-tuning, we use the following hyperparameter values:

Hyperparameter	Values
Model Size	{10M, 20M, 50M}
Model Dimension/Layers	{128/2, 512/4, 512/8 }
Learning Rate	{5e-4, 1e-4, 5e-5}
Tokenizer	{“1 Digit”, “3 Digits”, “Continuous”}

Table D.3: Hyperparameter values for from-scratch language model training runs.

Hyperparameter	Values
Learning Rate	{5e-4, 1e-4, 5e-5}
LoRA Rank	{8, 16, 32}
Batch Size	{8, 16}

Table D.4: Hyperparameter values for fine-tuning language model training runs.

Task	Tokenization	MAE	Standard Error
Distances	1 Digit	0.007583	0.001822
Distances	3 Digits	0.007587	0.001873
Distances	Continuous	0.002345	0.000049
Eigen	1 Digit	0.843819	0.051417
Eigen	3 Digits	0.949008	0.056988
Eigen	Continuous	0.731731	0.044416
Energy (C)	1 Digit	0.305043	0.016945
Energy (C)	3 Digits	0.541269	0.029265
Energy (C)	Continuous	0.167922	0.009055
Energy (D)	1 Digit	0.029822	0.005926
Energy (D)	3 Digits	0.039592	0.006085
Energy (D)	Continuous	0.006789	0.001359
Product	1 Digit	1.824334	0.131004
Product	3 Digits	0.636723	0.055891
Product	Continuous	0.186717	0.006272
Sum	1 Digit	0.003840	0.000149
Sum	3 Digits	0.024948	0.006033
Sum	Continuous	0.005297	0.000159

Table D.5: MAE values for different tasks and tokenization methods. Standard errors are calculated from 200 data points from each task.

D.7 MAE NUMBERS WITH STANDARD ERRORS

Table D.5 shows a full table of MAE values for each task and tokenization method, including standard errors calculated over 200 different examples from each task.

D.8 RASP-L PROGRAMS FOR ALL PAIRWISE DISTANCES

In the following, we provide RASP-L code for outputting a pairwise distance matrix from a sequence of points in 3D. For simplicity, we restrict input numbers to single tokens and we also provide negative integers to index where each element is in both the input and the output. We use the NumPy RASP-L [255] and make use of basic RASP-L functions like `induct_kqv` and `where`. Notably, when there is output space to perform computations for all pairs and produce the results, the solution can be implemented very straightforwardly.

```
1 # Constants representing special markers
2 COMMA = 255
3 START = 254
4 def distance(x):
5     """
6     Compute all pairs of distances autoregressively for a sequence of
7     ↪ points in 3d (with negative indices separating the different
8     ↪ elements, and negative indices used to specify matrix elements
9     ↪ in the output).
10
11     Ex: input=[-1 2 3 4 -2 11 8 4 -3 4 6 6 254]
12         output=[ -1  -2 106 255  -1  -3  17 255  -2  -1 106 255  -2  -2
13                ↪ 0 255  -2  -3
14
15     57...]
16     """
17     past0 = has_seen(x, full(x, START)) # Determine whether past the
18     ↪ start marker or not
19
20     # in the output sequence (-i,-j), induct on those values to find the
21     ↪ (-i, x_i, y_i, z_i)
22     # and (-j, x_j, y_j, z_j) points to compare
23     x1 = induct_kqv(x, x, x, 1)
```

```

16 x2 = induct_kqv(x, x, x, 2)
17 x3 = induct_kqv(x, x, x, 3)
18
19 y1 = x1 - shift_right(x1, 1)
20 y2 = x2 - shift_right(x2, 1)
21 y3 = x3 - shift_right(x3, 1)
22
23 # Compute the squared Euclidean distance
24 d = y1**2 + y2**2 + y3**2
25
26 prev = shift_right(x, 1)
27
28 # Determine readiness to produce a comma
29 rdynext = (past0 & (x >= 0) & (prev < 0))
30
31 # we need to get the index a and index b that were most recently used
32 start_a = -1
33 start_b = -1
34 # if we have already seen one full entry, then we grab those values
35 # otherwise we grab the first two values
36 # Find the last occurrence of COMMA
37 idlast = lasts(x, full(x, COMMA), default=-1)
38
39 # Select indices based on the last COMMA position
40 aselected = index_select(x, idlast - 3, default=0)
41 a = alast = where((idlast == -1) | (aselected == 0), full(x,
    ↪ start_a), aselected)
42
43 bselected = index_select(x, idlast - 2, default=0)
44 b = blast = where((idlast == -1) | (bselected == 0), full(x,
    ↪ start_b), bselected)

```

```

45     b -= 1
46
47     imax = minimum(x)
48
49     # Adjust indices based on the minimum value condition
50     a = where(b < imax, a - 1, a)
51     b = where(b < imax, full(x, -1), b)
52
53     out = d
54     # Determine readiness to produce the next negative index
55     rdynextnext = (x < 0) & (prev >= 0)
56     out = where(rdynextnext, b, out)
57     out = where(rdynext, full(x, COMMA), out)
58
59     out = where((x == COMMA) | (x == START), a, out)
60
61     return out

```

Listing D.1: Python code to compute all pairs of distances for a sequence of points

BIBLIOGRAPHY

- [1] Josh Abramson et al. “Accurate structure prediction of biomolecular interactions with AlphaFold 3”. In: *Nature* (2024), pp. 1–3.
- [2] Joshua Ainslie et al. “Colt5: Faster long-range transformers with conditional computation”. In: *arXiv preprint arXiv:2303.09752* (2023).
- [3] Nawaf Alampara, Santiago Miret, and Kevin Maik Jablonka. “MatText: Do Language Models Need More than Text & Scale for Materials Modeling?” In: *arXiv preprint arXiv:2406.17295* (2024).
- [4] Zeyuan Allen-Zhu and Yuanzhi Li. “Physics of Language Models: Part 1, Context-Free Grammar”. In: *arXiv preprint arXiv:2305.13673* (2023).
- [5] Victor M Martinez Alvarez, Rareş Roşca, and Cristian G Fălcuţescu. “DyNODE: Neural ordinary differential equations for dynamics modeling in continuous control”. In: *arXiv preprint arXiv:2009.04278* (2020).
- [6] Brandon Amos et al. “On the model-based stochastic value gradient for continuous reinforcement learning”. In: *Learning for Dynamics and Control*. PMLR. 2021, pp. 6–20.
- [7] Cem Anil et al. “Path Independent Equilibrium Models Can Better Exploit Test-Time Computation”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 7796–7809.

- [8] Rohan Anil et al. “Palm 2 technical report”. In: *arXiv preprint arXiv:2305.10403* (2023).
- [9] Abdul Fatir Ansari et al. “Chronos: Learning the language of time series”. In: *arXiv preprint arXiv:2403.07815* (2024).
- [10] Anthropic. “Introducing 100K Context Windows”. Anthropic blog. 2023.
- [11] Luis M Antunes, Keith T Butler, and Ricardo Grau-Crespo. “Crystal Structure Generation with Autoregressive Large Language Modeling”. In: *arXiv preprint arXiv:2307.04340* (2023).
- [12] Marianne Arriola et al. “Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models”. In: *arXiv preprint arXiv:2503.09573* (2025).
- [13] Ben Athiwaratkun et al. “There are many consistent explanations of unlabeled data: Why you should average”. In: *arXiv preprint arXiv:1806.05594* (2018).
- [14] Haim Avron and Sivan Toledo. “Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix”. In: *Journal of the ACM (JACM)* 58.2 (2011), pp. 1–34.
- [15] Aharon Azulay and Yair Weiss. “Why do deep convolutional networks generalize so poorly to small image transformations?” In: *arXiv preprint arXiv:1805.12177* (2018).
- [16] Stephen Balaban. “NVIDIA A100 GPU Benchmarks for Deep Learning”. Lambda Labs Blog. 2020.
- [17] Hangbo Bao, Li Dong, and Furu Wei. “BEiT: BERT Pre-Training of Image Transformers”. In: *arXiv preprint arXiv:2106.08254* (2021).
- [18] Simon Batzner et al. “E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials”. In: *Nature communications* 13.1 (2022), pp. 1–11.

- [19] Irwan Bello et al. “Revisiting resnets: Improved training and scaling strategies”. In: *arXiv preprint arXiv:2103.07579* (2021).
- [20] Gregory Benton et al. “Deep Probabilistic Time Series Forecasting over Long Horizons”. In: *openreview preprint* (2022).
- [21] Stella Biderman et al. “Emergent and Predictable Memorization in Large Language Models”. In: *arXiv preprint arXiv:2304.11158* (2023).
- [22] Aleksandar Botev et al. “Which priors matter? Benchmarking models for learning latent dynamics”. In: (2021).
- [23] Diane Bouchacourt, Mark Ibrahim, and Ari Morcos. “Grounding inductive biases in natural images: invariance stems from variations in data”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 19566–19579.
- [24] George EP Box and Gwilym M Jenkins. “Some recent advances in forecasting and control”. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 17.2 (1968), pp. 91–109.
- [25] Andres M Bran et al. “ChemCrow: Augmenting large-language models with chemistry tools”. In: *arXiv preprint arXiv:2304.05376* (2023).
- [26] Johann Brehmer et al. “Does equivariance matter at scale?” In: *arXiv preprint arXiv:2410.23179* (2024).
- [27] Andrew Brock et al. “High-performance large-scale image recognition without normalization”. In: *arXiv preprint arXiv:2102.06171* (2021).
- [28] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).
- [29] Michael M Bronstein et al. “Geometric deep learning: going beyond euclidean data”. In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

- [30] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.
- [31] Sébastien Bubeck et al. “Sparks of artificial general intelligence: Early experiments with gpt-4”. In: *arXiv preprint arXiv:2303.12712* (2023).
- [32] Andrew Campbell et al. “Generative flows on discrete state-spaces: Enabling multi-modal flows with applications to protein co-design”. In: *arXiv preprint arXiv:2402.04997* (2024).
- [33] Cristian Challu et al. “N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting”. In: *arXiv preprint arXiv:2201.12886* (2022).
- [34] Kent K Chang et al. “Speak, memory: An archaeology of books known to chatgpt/gpt-4”. In: *arXiv preprint arXiv:2305.00118* (2023).
- [35] François Charton. “Linear algebra with transformers”. In: *arXiv preprint arXiv:2112.01898* (2021).
- [36] Juan Manuel Zambrano Chaves et al. “Tx-LLM: A Large Language Model for Therapeutics”. In: *arXiv preprint arXiv:2406.06316* (2024).
- [37] Chi Chen and Shyue Ping Ong. “A universal graph deep learning interatomic potential for the periodic table”. In: *Nature Computational Science* 2.11 (2022), pp. 718–728.
- [38] Chun-Fu Chen, Quanfu Fan, and Rameswar Panda. “Crossvit: Cross-attention multi-scale vision transformer for image classification”. In: *arXiv preprint arXiv:2103.14899* (2021).
- [39] Ricky TQ Chen, Brandon Amos, and Maximilian Nickel. “Learning neural event functions for ordinary differential equations”. In: *arXiv preprint arXiv:2011.03902* (2020).
- [40] Sanyuan Chen et al. “VALL-E 2: Neural Codec Language Models are Human Parity Zero-Shot Text to Speech Synthesizers”. In: *arXiv preprint arXiv:2406.05370* (2024).

- [41] Tian Qi Chen et al. “Neural ordinary differential equations”. In: *Advances in neural information processing systems*. 2018, pp. 6571–6583.
- [42] Xiaokang Chen et al. *Janus-Pro: Unified Multimodal Understanding and Generation with Data and Model Scaling*. 2025.
- [43] Yunpeng Chen et al. *Dual Path Networks*. 2017.
- [44] Zhengdao Chen et al. “Symplectic recurrent neural networks”. In: *arXiv preprint arXiv:1909.13334* (2019).
- [45] François Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.
- [46] Paul F Christiano et al. “Deep reinforcement learning from human preferences”. In: *Advances in neural information processing systems* 30 (2017).
- [47] Xiangxiang Chu et al. “Twins: Revisiting the design of spatial attention in vision transformers”. In: *arXiv preprint arXiv:2104.13840* (2021).
- [48] Kurtland Chua et al. “Deep reinforcement learning in a handful of trials using probabilistic dynamics models”. In: *arXiv preprint arXiv:1805.12114* (2018).
- [49] Taco Cohen and Max Welling. “Group equivariant convolutional networks”. In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999.
- [50] Miles Cranmer. “Interpretable machine learning for science with PySR and SymbolicRegression.jl”. In: *arXiv preprint arXiv:2305.01582* (2023).
- [51] Miles Cranmer et al. “Lagrangian neural networks”. In: *arXiv preprint arXiv:2003.04630* (2020).

- [52] Ekin D Cubuk et al. “Randaugment: Practical automated data augmentation with a reduced search space”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 702–703.
- [53] Yue Cui, Jiandong Xie, and Kai Zheng. “Historical inertia: A neglected but powerful baseline for long sequence time-series forecasting”. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 2965–2969.
- [54] Stéphane d’Ascoli et al. “Convit: Improving vision transformers with soft convolutional inductive biases”. In: *arXiv preprint arXiv:2103.10697* (2021).
- [55] Fengyuan Dai et al. “Toward de novo protein design from natural language”. In: *bioRxiv* (2024), pp. 2024–08.
- [56] Zihang Dai et al. “CoAtNet: Marrying Convolution and Attention for All Data Sizes”. In: *arXiv preprint arXiv:2106.04803* (2021).
- [57] Sumanth Dathathri et al. “Plug and play language models: A simple approach to controlled text generation”. In: *arXiv preprint arXiv:1912.02164* (2019).
- [58] Grégoire Delétang et al. “Language Modeling Is Compression”. In: *arXiv preprint arXiv:2309.10668* (2023).
- [59] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
- [60] Shaan Desai et al. “Port-Hamiltonian Neural Networks for Learning Explicit Time-Dependent Dynamical Systems”. In: *arXiv preprint arXiv:2107.08024* (2021).
- [61] Tim Dettmers et al. “8-bit Optimizers via Block-wise Quantization”. In: *9th International Conference on Learning Representations, ICLR* (2022).

- [62] Xiaohan Ding et al. “Repvvg: Making vgg-style convnets great again”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13733–13742.
- [63] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).
- [64] Dazhao Du, Bing Su, and Zhewei Wei. “Preformer: Predictive Transformer with Multi-Scale Segment-wise Correlations for Long-Term Time Series Forecasting”. In: *arXiv preprint arXiv:2202.11356* (2022).
- [65] Haonan Duan et al. “Boosting the Predictive Power of Protein Representations with a Corpus of Text Annotations”. In: *bioRxiv* (2024), pp. 2024–07.
- [66] Abhimanyu Dubey et al. “The llama 3 herd of models”. In: *arXiv preprint arXiv:2407.21783* (2024).
- [67] Logan Engstrom et al. *A rotation and a translation suffice: Fooling cnns with simple transformations*. 2018.
- [68] Logan Engstrom et al. “Exploring the landscape of spatial robustness”. In: *International conference on machine learning*. PMLR. 2019, pp. 1802–1811.
- [69] Marc Finzi, Ke Alexander Wang, and Andrew Gordon Wilson. “Simplifying hamiltonian and lagrangian neural networks via explicit constraints”. In: *arXiv preprint arXiv:2010.13581* (2020).
- [70] Marc Finzi, Max Welling, and Andrew Gordon Wilson. “A Practical Method for Constructing Equivariant Multilayer Perceptrons for Arbitrary Matrix Groups”. In: *arXiv preprint arXiv:2104.09459* (2021).

- [71] Marc Finzi et al. “Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3165–3176.
- [72] Daniel Flam-Shepherd and Alán Aspuru-Guzik. “Language models can generate molecules, materials, and protein binding sites directly in three dimensions as XYZ, CIF, and PDB files”. In: *arXiv preprint arXiv:2305.05708* (2023).
- [73] Daniel Flam-Shepherd, Kevin Zhu, and Alán Aspuru-Guzik. “Atom-by-atom protein generation and beyond with language models”. In: *arXiv preprint arXiv:2308.09482* (2023).
- [74] Nathan Frey et al. *Neural Scaling of Deep Chemical Models*. 2022.
- [75] Nathan C Frey et al. “Neural scaling of deep chemical models”. In: *Nature Machine Intelligence* 5.11 (2023), pp. 1297–1305.
- [76] Philip Gage. “A new algorithm for data compression”. In: *The C Users Journal archive* 12 (1994), pp. 23–38.
- [77] Shanghua Gao et al. “Res2net: A new multi-scale backbone architecture”. In: *IEEE transactions on pattern analysis and machine intelligence* (2019).
- [78] Jacob R Gardner et al. “GPYtorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration”. In: *Advances in Neural Information Processing Systems*. 2018.
- [79] Federico Garza et al. “StatsForecast: Lightning fast forecasting with statistical and econometric models”. In: *PyCon: Salt Lake City, UT, USA* (2022).
- [80] Arnab Ghosh et al. “Steer: Simple temporal regularization for neural odes”. In: *arXiv preprint arXiv:2006.10711* (2020).

- [81] Rakshitha Godahewa et al. “Monash time series forecasting archive”. In: *arXiv preprint arXiv:2105.06643* (2021).
- [82] Micah Goldblum et al. “The No Free Lunch Theorem, Kolmogorov Complexity, and the Role of Inductive Biases in Machine Learning”. In: *arXiv preprint arXiv:2304.05366* (2023).
- [83] Siavash Golkar et al. “xval: A continuous number encoding for large language models”. In: *arXiv preprint arXiv:2310.02989* (2023).
- [84] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [85] Samuel J Greydanus, Misko Dzumba, and Jason Yosinski. “Hamiltonian neural networks”. In: (2019).
- [86] Nate Gruver et al. “Fine-Tuned Language Models Generate Stable Inorganic Materials as Text”. In: *arXiv preprint arXiv:2402.04379* (2024).
- [87] Nate Gruver et al. “Large Language Models Are Zero-Shot Time Series Forecasters”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.
- [88] Nate Gruver et al. “Large language models are zero-shot time series forecasters”. In: *Advances in Neural Information Processing Systems* 36 (2024).
- [89] Nate Gruver et al. “Protein Design with Guided Discrete Diffusion”. In: *arXiv preprint arXiv:2305.20009* (2023).
- [90] Nate Gruver et al. “The lie derivative for measuring learned equivariance”. In: *arXiv preprint arXiv:2210.02984* (2022).
- [91] Jian Guo et al. “GluonCV and GluonNLP: Deep Learning in Computer Vision and Natural Language Processing”. In: *Journal of Machine Learning Research* 21.23 (2020), pp. 1–7.

- [92] Jayesh K Gupta et al. “A general framework for structured learning of mechanical systems”. In: *arXiv preprint arXiv:1902.08705* (2019).
- [93] Jayesh K Gupta et al. “Structured mechanical models for robot learning and control”. In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 328–337.
- [94] Tuomas Haarnoja et al. “Soft actor-critic algorithms and applications”. In: *arXiv preprint arXiv:1812.05905* (2018).
- [95] Jürgen Hafner. “Ab-initio simulations of materials using VASP: Density-functional theory and beyond”. In: *Journal of computational chemistry* 29.13 (2008), pp. 2044–2078.
- [96] Brian C Hall. “Lie groups, Lie algebras, and representations”. In: *Quantum Theory for Mathematicians*. Springer, 2013, pp. 333–366.
- [97] Dongyoon Han et al. “Rethinking Channel Dimensions for Efficient Model Design”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 732–741.
- [98] Kai Han et al. “Transformer in transformer”. In: *arXiv preprint arXiv:2103.00112* (2021).
- [99] Thomas Hayes et al. “Simulating 500 million years of evolution with a language model”. In: *Science* (2025), eads0018.
- [100] Tomas Hayes et al. “Simulating 500 million years of evolution with a language model”. In: *bioRxiv* (2024), pp. 2024–07.
- [101] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015.
- [102] Kaiming He et al. “Masked Autoencoders Are Scalable Vision Learners”. In: *arXiv:2111.06377* (2021).

- [103] Tong He et al. “Bag of Tricks for Image Classification with Convolutional Neural Networks”. In: *arXiv preprint arXiv:1812.01187* (2018).
- [104] Stefan Hegselmann et al. “Tabllm: Few-shot classification of tabular data with large language models”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2023, pp. 5549–5581.
- [105] Dan Hendrycks et al. “Measuring massive multitask language understanding”. In: *arXiv preprint arXiv:2009.03300* (2020).
- [106] Byeongho Heo et al. “Rethinking spatial dimensions of vision transformers”. In: *arXiv preprint arXiv:2103.16302* (2021).
- [107] Julien Herzen et al. “Darts: User-friendly modern machine learning for time series”. In: *The Journal of Machine Learning Research* 23.1 (2022), pp. 5442–5447.
- [108] Hansika Hewamalage, Klaus Ackermann, and Christoph Bergmeir. “Forecast evaluation for data scientists: common pitfalls and best practices”. In: *Data Mining and Knowledge Discovery* 37.2 (2023), pp. 788–832.
- [109] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.
- [110] Andreas Hochlehnert et al. “Learning contact dynamics using physically structured neural networks”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 2152–2160.
- [111] Ari Holtzman et al. “The curious case of neural text degeneration”. In: *arXiv preprint arXiv:1904.09751* (2019).
- [112] J. Edward Hu et al. “LoRA: Low-Rank Adaptation of Large Language Models”. In: *ArXiv abs/2106.09685* (2021).

- [113] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-excitation networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141.
- [114] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.
- [115] Kexin Huang et al. “Automated Hypothesis Validation with Agentic Sequential Falsifications”. In: *arXiv preprint arXiv:2502.09858* (2025).
- [116] Michael J Hutchinson et al. “Lietransformer: Equivariant self-attention for lie groups”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 4533–4543.
- [117] Rob J Hyndman et al. *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media, 2008.
- [118] Anubhav Jain et al. “Commentary: The Materials Project: A materials genome approach to accelerating materials innovation”. In: *APL materials* 1.1 (2013).
- [119] Michael Janner et al. “When to trust your model: Model-based policy optimization”. In: *arXiv preprint arXiv:1906.08253* (2019).
- [120] Rui Jiao et al. “Crystal Structure Prediction by Joint Equivariant Diffusion on Lattices and Fractional Coordinates”. In: *Workshop on “Machine Learning for Materials” ICLR 2023*. 2023.
- [121] Pengzhan Jin et al. “SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems”. In: *Neural Networks* 132 (2020), pp. 166–179.
- [122] John Jumper et al. “Highly accurate protein structure prediction with AlphaFold”. In: *Nature* 596.7873 (2021), pp. 583–589.
- [123] Kaggle and EyePacs. *Kaggle Diabetic Retinopathy Detection*. 2015.

- [124] Sanyam Kapoor et al. “Large Language Models Must Be Taught to Know What They Don’t Know”. In: *arXiv preprint arXiv:2406.08391* (2024).
- [125] George Em Karniadakis et al. “Physics-informed machine learning”. In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.
- [126] Tero Karras et al. “Alias-free generative adversarial networks”. In: *Advances in Neural Information Processing Systems* 34 (2021).
- [127] Jakob Nikolas Kather et al. “Multi-class texture analysis in colorectal cancer histology”. In: *Scientific reports* 6 (2016), p. 27988.
- [128] Shruti Kaushik et al. “AI in healthcare: time-series forecasting using statistical, neural, and ensemble architectures”. In: *Frontiers in big data* 3 (2020), p. 4.
- [129] Scott Kirklin et al. “The Open Quantum Materials Database (OQMD): assessing the accuracy of DFT formation energies”. In: *npj Computational Materials* 1.1 (2015), pp. 1–15.
- [130] Guokun Lai et al. “Modeling long-and short-term temporal patterns with deep neural networks”. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 2018, pp. 95–104.
- [131] Samuli Laine and Timo Aila. “Temporal ensembling for semi-supervised learning”. In: *arXiv preprint arXiv:1610.02242* (2016).
- [132] Janice Lan et al. “AdsorbML: Accelerating Adsorption Energy Calculations with Machine Learning”. In: *arXiv preprint arXiv:2211.16486* (2022).
- [133] Dmitry Laptev et al. “Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 289–297.

- [134] Colin Lea et al. “Temporal convolutional networks: A unified approach to action segmentation”. In: *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*. Springer. 2016, pp. 47–54.
- [135] Yann LeCun, Yoshua Bengio, et al. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.
- [136] Yann LeCun et al. “Handwritten digit recognition with a back-propagation network”. In: *Advances in neural information processing systems* 2 (1989).
- [137] Katherine Lee et al. “Deduplicating training data makes language models better”. In: *arXiv preprint arXiv:2107.06499* (2021).
- [138] Kookjin Lee, Nathaniel Trask, and Panos Stinis. “Machine learning structure preserving brackets for forecasting irreversible processes”. In: *Advances in Neural Information Processing Systems* 34 (2021).
- [139] Benedict J Leimkuhler and Robert D Skeel. “Symplectic numerical integrators in constrained Hamiltonian systems”. In: *Journal of Computational Physics* 112.1 (1994), pp. 117–125.
- [140] Karel Lenc and Andrea Vedaldi. “Understanding image representations by measuring their equivariance and equivalence”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 991–999.
- [141] Shuo-Hui Li et al. “Neural canonical transformation with symplectic flows”. In: *Physical Review X* 10.2 (2020), p. 021020.
- [142] Xiang Li et al. “Selective kernel networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 510–519.

- [143] Ziming Li et al. “Machine-Learning Non-Conservative Dynamics for New-Physics Detection”. In: *arXiv preprint arXiv:2106.00026* (2021).
- [144] Weixin Liang et al. “Can large language models provide useful feedback on research papers? A large-scale empirical analysis”. In: *NEJM AI* 1.8 (2024), AIoa2400196.
- [145] Hanxiao Liu et al. “Pay Attention to MLPs”. In: *arXiv preprint arXiv:2105.08050* (2021).
- [146] Tiedong Liu and Bryan Kian Hsiang Low. “Goat: Fine-tuned LLaMA Outperforms GPT-4 on Arithmetic Tasks”. In: *arXiv preprint arXiv:2305.14201* (2023).
- [147] Yinhan Liu et al. “Multilingual Denoising Pre-training for Neural Machine Translation”. In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 726–742.
- [148] Ze Liu et al. “Swin transformer: Hierarchical vision transformer using shifted windows”. In: *arXiv preprint arXiv:2103.14030* (2021).
- [149] Ziming Liu et al. “Physics-Augmented Learning: A New Paradigm Beyond Physics-Informed Learning”. In: *arXiv preprint arXiv:2109.13901* (2021).
- [150] Michael Lutter, Christian Ritter, and Jan Peters. “Deep lagrangian networks: Using physics as model prior for deep learning”. In: *arXiv preprint arXiv:1907.04490* (2019).
- [151] Michael Lutter, Christian Ritter, and Jan Peters. “Deep lagrangian networks: Using physics as model prior for deep learning”. In: *arXiv preprint arXiv:1907.04490* (2019).
- [152] Dhruv Mahajan et al. “Exploring the limits of weakly supervised pretraining”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 181–196.
- [153] Dhruv Kumar Mahajan et al. “Exploring the Limits of Weakly Supervised Pretraining”. In: *ECCV*. 2018.

- [154] Diego Marcos et al. “Rotation equivariant vector field networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5048–5057.
- [155] Sean McLeish et al. “Transformers Can Do Arithmetic with the Right Embeddings”. In: *arXiv preprint arXiv:2405.17399* (2024).
- [156] Dushyant Mehta et al. “XNect”. In: *ACM Transactions on Graphics* 39.4 (2020). ISSN: 1557-7368. DOI: [10.1145/3386569.3392410](https://doi.org/10.1145/3386569.3392410).
- [157] William Merrill and Ashish Sabharwal. “The parallelism tradeoff: Limitations of log-precision transformers”. In: *Transactions of the Association for Computational Linguistics* 11 (2023), pp. 531–545.
- [158] Tomas Mikolov et al. “Recurrent neural network based language model.” In: *Inter-speech*. Vol. 2. 3. Makuhari. 2010, pp. 1045–1048.
- [159] Suvir Mirchandani et al. “Large language models as general pattern machines”. In: *arXiv preprint arXiv:2307.04721* (2023).
- [160] Seungwhan Moon et al. “Anymal: An efficient and scalable any-modality augmented language model”. In: *arXiv preprint arXiv:2309.16058* (2023).
- [161] Steffen Moritz and Thomas Bartz-Beielstein. “imputeTS: time series missing value imputation in R.” In: *R J.* 9.1 (2017), p. 207.
- [162] Allen Nie et al. “EVOLvE: Evaluating and Optimizing LLMs For Exploration”. In: *arXiv preprint arXiv:2410.06238* (2024).
- [163] Alaaeldin El-Nouby et al. “XCiT: Cross-Covariance Image Transformers”. In: *arXiv preprint arXiv:2106.09681* (2021).
- [164] Maxwell Nye et al. “Show your work: Scratchpads for intermediate computation with language models”. In: *arXiv preprint arXiv:2112.00114* (2021).

- [165] Chris Olah et al. “Naturally occurring equivariance in neural networks”. In: *Distill* 5.12 (2020), e00024–004.
- [166] Catherine Olsson et al. “In-context Learning and Induction Heads”. In: *Transformer Circuits Thread* (2022). <https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html>.
- [167] Shyue Ping Ong et al. “Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis”. In: *Computational Materials Science* 68 (2013), pp. 314–319.
- [168] Aaron van den Oord et al. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).
- [169] OpenAI. “GPT-4 technical report”. In: *arXiv* (2023).
- [170] Boris N Oreshkin et al. “N-BEATS: Neural basis expansion analysis for interpretable time series forecasting”. In: *Journal of Machine Learning Research* 21.111 (2020), pp. 1–63.
- [171] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744.
- [172] Feiyang Pan et al. “Trust the Model When It Is Confident: Masked Model-based Actor-Critic”. In: *arXiv preprint arXiv:2010.04893* (2020).
- [173] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems* 32. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035.
- [174] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Neural Information Processing Systems*. 2019.

- [175] Liudmila Prokhorenkova et al. “CatBoost: unbiased boosting with categorical features”. In: *Advances in Neural Information Processing Systems*. Vol. 31. NeurIPS. 2018, pp. 6638–6648.
- [176] Alec Radford et al. “Learning transferable visual models from natural language supervision”. In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.
- [177] Ilija Radosavovic et al. “Designing network design spaces”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10428–10436.
- [178] Maithra Raghu et al. *Do Vision Transformers See Like Convolutional Neural Networks?* 2021.
- [179] Raghunathan Ramakrishnan et al. “Quantum chemistry structures and properties of 134 kilo molecules”. In: *Scientific data* 1.1 (2014), pp. 1–7.
- [180] Scott Reed et al. “A generalist agent”. In: *arXiv preprint arXiv:2205.06175* (2022).
- [181] Antônio H Ribeiro and Thomas B Schön. “How convolutional neural networks deal with aliasing”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 2755–2759.
- [182] James E Saal et al. “Materials design and discovery with high-throughput density functional theory: the open quantum materials database (OQMD)”. In: *Jom* 65 (2013), pp. 1501–1509.
- [183] Subham Sahoo et al. “Simple and effective masked diffusion language models”. In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 130136–130184.
- [184] David Salinas et al. “DeepAR: Probabilistic forecasting with autoregressive recurrent networks”. In: *International Journal of Forecasting* 36.3 (2020), pp. 1181–1191.

- [185] Guillaume Sanchez et al. “Stay on topic with Classifier-Free Guidance”. In: *arXiv preprint arXiv:2306.17806* (2023).
- [186] Mark Sandler et al. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.
- [187] Victor Garcia Satorras, Emiel Hooeboom, and Max Welling. “E (n) equivariant graph neural networks”. In: *arXiv preprint arXiv:2102.09844* (2021).
- [188] Philipp Schmid. *LLaMA 2 on Amazon Sagemaker, a Benchmark*. <https://huggingface.co/blog/llama-sagemaker-benchmark>. 2023.
- [189] Avi Schwarzschild et al. “Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 6695–6706.
- [190] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [191] Nima Shoghi et al. “From molecules to materials: Pre-training large generalizable models for atomic property prediction”. In: *arXiv preprint arXiv:2310.16802* (2023).
- [192] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).
- [193] Samuel Stanton et al. “Accelerating bayesian optimization for biological sequence design with denoising autoencoders”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 20459–20478.
- [194] Ke Sun et al. *High-Resolution Representations for Labeling Pixels and Regions*. 2019.
- [195] Wenhao Sun et al. “The thermodynamic scale of inorganic crystalline metastability”. In: *Science advances* 2.11 (2016), e1600225.

- [196] Ilya Sutskever. “An observation on Generalization”. Workshop on Large Language Models and Transformers. 2023.
- [197] Christian Szegedy et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016.
- [198] Mingxing Tan and Quoc Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6105–6114.
- [199] Mingxing Tan and Quoc V Le. “Efficientnetv2: Smaller models and faster training”. In: *arXiv preprint arXiv:2104.00298* (2021).
- [200] Mingxing Tan and Quoc V. Le. *MixConv: Mixed Depthwise Convolutional Kernels*. 2019.
- [201] Mingxing Tan et al. *MnasNet: Platform-Aware Neural Architecture Search for Mobile*. 2019.
- [202] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.
- [203] Ilya Tolstikhin et al. “Mlp-mixer: An all-mlp architecture for vision”. In: *arXiv preprint arXiv:2105.01601* (2021).
- [204] Yunjin Tong et al. “Symplectic neural networks in Taylor series form for Hamiltonian systems”. In: *Journal of Computational Physics* 437 (2021), p. 110325.
- [205] Hugo Touvron et al. “Going deeper with image transformers”. In: *arXiv preprint arXiv:2103.17239* (2021).
- [206] Hugo Touvron et al. “Llama 2: Open Foundation and Fine-Tuned Chat Models”. In: *ArXiv abs/2307.09288* (2023).

- [207] Hugo Touvron et al. “LLaMA: Open and Efficient Foundation Language Models”. In: *ArXiv abs/2302.13971* (2023).
- [208] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).
- [209] Hugo Touvron et al. “Resmlp: Feedforward networks for image classification with data-efficient training”. In: *arXiv preprint arXiv:2105.03404* (2021).
- [210] Hugo Touvron et al. “Training data-efficient image transformers & distillation through attention”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10347–10357.
- [211] Juan R Trapero, Nikolaos Kourentzes, and Robert Fildes. “On the identification of sales forecasting models in the presence of promotions”. In: *Journal of the Operational Research Society* 66.2 (2015), pp. 299–307.
- [212] Asher Trockman and J Zico Kolter. “Patches are all you need?” In: *arXiv preprint arXiv:2201.09792* (2022).
- [213] Cristina Vasconcelos et al. “Impact of aliasing on generalization in deep convolutional networks”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10529–10538.
- [214] Bastiaan S Veeling et al. “Rotation equivariant cnns for digital pathology”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2018, pp. 210–218.
- [215] Eric Wallace et al. “Universal adversarial triggers for attacking and analyzing NLP”. In: *arXiv preprint arXiv:1908.07125* (2019).
- [216] Chien-Yao Wang et al. *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2019.

- [217] Tingwu Wang and Jimmy Ba. “Exploring model-based planning with policy networks”. In: *arXiv preprint arXiv:1906.08649* (2019).
- [218] Xinlong Wang et al. *Emu3: Next-Token Prediction is All You Need*. 2024.
- [219] Logan Ward et al. “Matminer: An open source toolkit for materials data mining”. In: *Computational Materials Science* 152 (2018), pp. 60–69.
- [220] Jason Wei et al. “Chain of thought prompting elicits reasoning in large language models”. In: *arXiv preprint arXiv:2201.11903* (2022).
- [221] Jason Wei et al. “Finetuned language models are zero-shot learners”. In: *arXiv preprint arXiv:2109.01652* (2021).
- [222] Maurice Weiler and Gabriele Cesa. “General E(2)-Equivariant Steerable CNNs”. In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2019.
- [223] Andrew D White. “The future of chemistry is language”. In: *Nature Reviews Chemistry* (2023), pp. 1–2.
- [224] Ross Wightman. *PyTorch Image Models*. <https://github.com/rwightman/pytorch-image-models>. 2019. DOI: [10.5281/zenodo.4414861](https://doi.org/10.5281/zenodo.4414861).
- [225] Ross Wightman, Hugo Touvron, and Hervé Jégou. “Resnet strikes back: An improved training procedure in timm”. In: *arXiv preprint arXiv:2110.00476* (2021).
- [226] Andrew Wilson and Ryan Adams. “Gaussian process kernels for pattern discovery and extrapolation”. In: *International conference on machine learning*. PMLR. 2013, pp. 1067–1075.
- [227] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45.

- [228] Daniel E Worrall et al. “Harmonic networks: Deep translation and rotation equivariance”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5028–5037.
- [229] Felix Wu et al. “Simplifying graph convolutional networks”. In: *International conference on machine learning*. PMLR. 2019, pp. 6861–6871.
- [230] Haixu Wu et al. “Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting”. In: *Advances in Neural Information Processing Systems* 34 (2021).
- [231] Saining Xie et al. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.
- [232] Tian Xie et al. “Crystal diffusion variational autoencoder for periodic material generation”. In: *arXiv preprint arXiv:2110.06197* (2021).
- [233] Shiyong Xiong et al. “Nonseparable symplectic neural networks”. In: *arXiv preprint arXiv:2010.12636* (2020).
- [234] Hao Xue and Flora D. Salim. *PromptCast: A New Prompt-based Learning Paradigm for Time Series Forecasting*. 2023.
- [235] I. Zeki Yalniz et al. “Billion-scale semi-supervised learning for image classification”. In: *CoRR* abs/1905.00546 (2019).
- [236] Mengjiao Yang et al. “Scalable Diffusion for Materials Generation”. In: *arXiv preprint arXiv:2311.09235* (2023).
- [237] Fisher Yu et al. *Deep Layer Aggregation*. 2019.
- [238] Lijun Yu et al. “Language Model Beats Diffusion—Tokenizer is Key to Visual Generation”. In: *arXiv preprint arXiv:2310.05737* (2023).

- [239] Zheng Yuan et al. “How well do Large Language Models perform in Arithmetic tasks?” In: *arXiv preprint arXiv:2304.02015* (2023).
- [240] Sangdoon Yun et al. “Cutmix: Regularization strategy to train strong classifiers with localizable features”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6023–6032.
- [241] Ailing Zeng et al. “Are transformers effective for time series forecasting?” In: *arXiv preprint arXiv:2205.13504* (2022).
- [242] Claudio Zeni et al. “MatterGen: a generative model for inorganic materials design”. In: *arXiv preprint arXiv:2312.03687* (2023).
- [243] Hang Zhang et al. “ResNeSt: Split-Attention Networks”. In: *arXiv preprint arXiv:2004.08955* (2020).
- [244] Hongyi Zhang et al. “mixup: Beyond empirical risk minimization”. In: *arXiv preprint arXiv:1710.09412* (2017).
- [245] Richard Zhang. “Making convolutional networks shift-invariant again”. In: *International conference on machine learning*. PMLR. 2019, pp. 7324–7334.
- [246] Tianyi Zhang et al. *QPyTorch: A Low-Precision Arithmetic Simulation Framework*. 2019.
- [247] Zhi Zhang et al. “Bag of Freebies for Training Object Detection Neural Networks”. In: *arXiv preprint arXiv:1902.04103* (2019).
- [248] Zizhao Zhang et al. “Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 2022, pp. 3417–3425.

- [249] Artem Zhohus et al. “BindGPT: A Scalable Framework for 3D Molecular Design via Language Modeling and Reinforcement Learning”. In: *arXiv preprint arXiv:2406.03686* (2024).
- [250] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. “Benchmarking Energy-Conserving Neural Networks for Learning Dynamics from Data”. In: *Learning for Dynamics and Control*. PMLR. 2021, pp. 1218–1229.
- [251] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. “Dissipative symoden: Encoding hamiltonian dynamics with dissipation and control into deep learning”. In: *arXiv preprint arXiv:2002.08860* (2020).
- [252] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. “Extending Lagrangian and Hamiltonian Neural Networks with Differentiable Contact Models”. In: *Advances in Neural Information Processing Systems* 34 (2021).
- [253] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. “Symplectic ode-net: Learning hamiltonian dynamics with control”. In: *arXiv preprint arXiv:1909.12077* (2019).
- [254] Haoyi Zhou et al. “Informer: Beyond efficient transformer for long sequence time-series forecasting”. In: *Proceedings of AAAI*. 2021.
- [255] Hattie Zhou et al. “What algorithms can transformers learn? a study in length generalization”. In: *arXiv preprint arXiv:2310.16028* (2023).
- [256] Tian Zhou et al. “FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting”. In: *Proc. 39th International Conference on Machine Learning (ICML 2022)*. Baltimore, Maryland, 2022.
- [257] Yanzhao Zhou et al. “Oriented response networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 519–528.

- [258] Zhaocheng Zhu et al. “Large language models can learn rules”. In: *arXiv preprint arXiv:2310.07064* (2023).