

Multilevel and Multimodal Transformers for Text, Vision, and Scientific Discovery

by

Yanming Kang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Applied Mathematics

Waterloo, Ontario, Canada, 2026

© Yanming Kang 2026

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

- External Examiner: Eran Treister
Associate Professor, Stein Faculty of Computer
and Information Science
Ben Gurion University of the Negev
- Supervisors: Hans De Sterck
Professor, Applied Mathematics, University of Waterloo
Giang Tran
Associate Professor, Applied Mathematics,
University of Waterloo
- Internal Member: Mohammad Kohandel
Professor, Applied Mathematics, University of Waterloo
- Internal Member: Jun Liu
Professor, Applied Mathematics, University of Waterloo
- Internal-External Member: Ali Ghodsi
Professor, Department of Statistics and Actuarial Science

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The research presented in Chapters 3 and 4 was conducted collaboratively. Individual contributions are detailed below.

Chapter 3: Fast Multipole Attention (FMA).

Yanming Kang, Dr. Hans De Sterck, and Dr. Giang Tran jointly designed the method. Yanming Kang implemented the method and conducted the experiments.

Chapter 4: MOSAIC-SR.

Yanming Kang, Peiyi Zheng, Dr. Hans De Sterck, and Dr. Giang Tran jointly designed the model and method. Yanming Kang and Peiyi Zheng jointly implemented the method. Peiyi Zheng conducted the evaluations on SRSD-Feynman and prepared the visualizations. Yanming Kang conducted the evaluations on the other datasets and the ablation studies.

Chapter 4: RIMS.

Yanming Kang, Peiyi Zheng, Dr. Hans De Sterck, and Dr. Giang Tran jointly developed the method and its theoretical results. Yanming Kang implemented the method and conducted the experiments on the AI Feynman datasets.

Abstract

This thesis presents two projects on Transformer architectures and their use in text, vision, and scientific discovery. The first develops efficient multilevel attention for text and vision. The second develops Transformer-based symbolic regression together with a method for selecting among candidate equations.

The first project focuses on the computational cost of self attention in Transformers. Standard self attention compares all pairs of tokens, which gives a global receptive field but scales quadratically with sequence length or image resolution. We propose an efficient attention mechanism, called Fast Multipole Attention (FMA), that adapts ideas from the Fast Multipole Method, a classical algorithm originally developed for large n -body simulations. The main idea of our approach is that nearby tokens interact at full resolution, while distant tokens interact through learned summaries arranged in a multilevel hierarchy. In one spatial dimension, the construction yields an efficient attention mechanism for autoregressive and bidirectional language modeling. In two spatial dimensions, a quadtree variant extends the same idea to image classification and semantic segmentation. The resulting models preserve global interaction while reducing attention cost to nearly linear or linear scaling with respect to the input size. While other multilevel and efficient Transformers have appeared in the literature, our approach is attractive because it learns summaries on multiple levels, is parsimonious and dimension independent in principle, and obtains strong empirical results.

The second project studies symbolic equation recovery and model selection. We propose MOSAIC-SR, a hybrid framework in which a pretrained Transformer proposes multiple symbolic sketches that initialize searches in promising regions of the expression space. Monte Carlo tree search maps decoded variable tokens to observed inputs, after which parallel local searches jointly repair symbolic structure and refit constants using scale-aware optimization. The method targets algebraic recovery rather than predictive fit alone. On SRSD-Feynman data with and without irrelevant dummy variables and on six additional benchmarks, MOSAIC-SR achieves the highest symbolic solution rate on every dataset while ranking among the top two methods in predictive accuracy. As a related model-selection method, we propose RIMS, a post hoc residual-invariance method for selecting among candidate symbolic equations in multi-environment data. RIMS addresses cases in which a low error expression depends on descendants of the target and fails to represent the underlying mechanism. It compares residual distributions across environments using a normalized Cramér-von Mises discrepancy. RIMS requires no retraining and applies to candidate pools from different symbolic regression systems. Theoretical results give a finite-candidate recovery guarantee under uniform estimation error and a detectable invariance

gap, and characterize invariant downstream channels as an identification boundary. On the 119 official AI Feynman datasets augmented with a planted measurement variable, default PySR selection recovered no physical mechanisms, whereas RIMS recovered 73 of the 81 mechanisms available in the union candidate pools.

Acknowledgements

I would like to thank my supervisors, Giang Tran and Hans De Sterck, for their guidance and support throughout my PhD. I would also like to thank Peiyi Zheng for her collaboration and help. I am grateful to my parents for their continued support. Finally, I would like to thank everyone who has helped and supported me along the way.

Table of Contents

Examining Committee Membership	ii
Author’s Declaration	iii
Statement of Contributions	iv
Abstract	v
Acknowledgements	vii
List of Figures	xii
List of Tables	xvi
List of Abbreviations	xx
List of Symbols	xxi
1 Introduction	1
1.1 The Transformer Architecture	2
1.2 Efficient Self-Attention	4
1.3 Learning-Based Symbolic Regression	5
1.4 Invariance in Symbolic Regression	7

1.5	Objectives and Contributions	8
1.6	Thesis Overview	10
2	Background and Preliminaries	12
2.1	Transformers and Applications	12
2.1.1	Transformer Architecture for One Dimensional Sequences	13
2.1.2	Vision Transformer Architecture and Visual Backbones	20
2.1.3	Pretraining and Downstream Adaptation	25
2.1.4	One Dimensional Sequence Applications	29
2.1.5	Vision and Imaging Applications	33
2.1.6	Multimodal Transformers and Contrastive Learning	35
2.2	Efficient Transformers and Attention	36
2.2.1	The Cost of Dense Global Attention	37
2.2.2	Sparse and Local Attention	38
2.2.3	Clustering and Routing Attention	39
2.2.4	Linearized Attention	40
2.2.5	Pooling and Downsampling	41
2.2.6	Hierarchical Attention	42
2.2.7	IO-Aware Exact Attention	45
2.3	Hierarchical and Multilevel Numerical Methods	46
2.3.1	All Pairs Sums and the Fast Multipole Method	46
2.3.2	Hierarchical Matrices and Low Rank Off Diagonal Structure	48
2.3.3	Tree Structures and Geometric Decomposition	50
2.3.4	What Carries Over to Fast Multipole Attention	51
2.4	Symbolic Regression	52
2.4.1	Problem Formulation and Evaluation	53
2.4.2	Classical Search-Based Symbolic Regression	55
2.4.3	Transformer Based Symbolic Regression	56

2.4.4	Other Neural Symbolic Regression Methods	59
2.4.5	Templates, Sketches, and Decomposition	61
2.4.6	Monte Carlo Tree Search for Symbolic Regression	62
2.5	Invariance and Causal Inference	64
2.5.1	Structural Causal Models and Interventions	64
2.5.2	Invariance Principles in Causal Learning	65
2.5.3	Descendant Shortcuts and the Failure of Pooled Risk	66
2.5.4	Two-Sample Tests and Distributional Discrepancy	67
2.5.5	Out-of-Distribution Generalization and Domain Shift	69
3	Fast Multipole Attention	70
3.1	Problem Setting and Design Goals	70
3.2	Fast Multipole Attention Method	74
3.2.1	Common Notation	74
3.2.2	Near Field and Far Field Decomposition	77
3.2.3	One Dimensional Construction	79
3.2.4	Two Dimensional Construction	80
3.2.5	Causal Constraints and Far Field Capacity	82
3.3	Complexity and Implementation	84
3.4	Relationship to Prior Attention Methods	88
3.5	Experimental Setup	92
3.6	Language Modeling Results	93
3.6.1	Autoregressive Language Modeling	93
3.6.2	Bidirectional Language Modeling	96
3.7	Vision Results	97
3.7.1	Image Classification	97
3.7.2	Semantic Segmentation	101
3.7.3	Object Detection	101

3.8	Ablations and Approximation Analysis	103
3.9	Symbol Retrieval	107
3.10	Discussion and Limitations	109
4	Symbolic Equation Recovery and Selection	111
4.1	Transformer Guided Symbolic Regression with MOSAIC-SR	111
4.1.1	Introduction	111
4.1.2	Method	113
4.1.3	Experiment Settings	117
4.1.4	SRSD-Feynman Results	122
4.1.5	Cross-Benchmark Results	134
4.1.6	Qualitative Recovery Across Taylor Orders	138
4.1.7	Validation NMSE Trajectories during Local Search	141
4.1.8	Ablation Studies	148
4.1.9	Discussion	149
4.2	Residual-Invariant Selection of Symbolic Models	151
4.2.1	Motivation and Scope	151
4.2.2	Problem Setting	152
4.2.3	Residual Invariant Model Selection	153
4.2.4	Finite-Candidate Guarantee and Identification Boundary	155
4.2.5	Experiments	157
4.2.6	Discussion	158
5	Conclusion	160
5.1	Summary of Contributions	160
5.2	Limitations and Open Problems	161
5.3	Future Directions	161
	References	162

List of Figures

2.1	Input construction for a Transformer. Token indices are mapped to rows of the embedding table. Position information is added to form the first hidden matrix.	14
2.2	Encoder–decoder Transformer data flow as in the original Transformer [159]. For simplicity, only one Transformer block is represented in each of the Encoder and Decoder. Each attention module contains query, key, and value projections before its heads and an output projection after the head outputs are concatenated. Encoder-only models retain the encoder stack, decoder-only models retain the masked decoder stack without encoder cross attention, and encoder–decoder models use both stacks.	19
2.3	Representative fixed sparse patterns. A sliding window produces a diagonal band; global tokens add full rows and columns; shifted spatial windows connect neighboring regions across consecutive layers.	39
2.4	Data-dependent routing attention. Similar queries and keys are assigned to common buckets, and dense attention is evaluated only within each bucket.	40
2.5	Linearized attention changes the multiplication order. Keys and values are first compressed into summaries S and z , which are then reused for every transformed query.	40
2.6	Pooling and downsampling reduce the key–value sequence before attention. The resulting interaction matrix is rectangular, with n queries and only k pooled keys.	41
2.7	Binary hierarchy in H-Transformer-1D. Nearby interactions are represented by fine tokens, while distant groups interact through coarser summaries.	42
2.8	Adaptive block refinement in multi-resolution attention. Most of the attention matrix remains coarse, while selected regions are subdivided and evaluated at finer resolution.	43

2.9	Near-field/far-field decomposition in FMMformer. A banded matrix retains local interactions, and low-rank factors communicate information over the full sequence.	44
2.10	IO-aware exact attention processes the score matrix tile by tile. Only small query, key, and value blocks and a running softmax state reside in fast on-chip memory, so the full $n \times n$ matrix is never written to external memory.	45
2.11	Tree-structured FMM computation. Source weights are aggregated upward into cluster moments. Well-separated clusters communicate through compressed far-field translations, which are propagated downward through the target tree. Nearby leaf interactions are evaluated directly.	48
2.12	Regular geometric partitions for one- and two dimensional data. A one dimensional interval is bisected into dyadic child intervals. A two dimensional square is divided into four children at each quadtree refinement. In both cases, every coarse cell is the union of its children.	50
3.1	Conceptual view of one dimensional FMA. Full attention computes a separate interaction between query token i and every source token. FMA keeps exact attention in a three-cell near field of width $3r$ and represents the far field by learned summaries of progressively larger groups.	75
3.2	Sparsified attention matrix for one dimensional FMA. White cells denote exact near field interactions. Gray blocks denote summarized far field interactions, with darker shades indicating coarser hierarchy levels.	76
3.3	FMA two dimensional hierarchy and a corresponding sparsified attention matrix. White cells denote exact near field interactions. Gray cells denote summarized far field interactions, with darker shades indicating coarser hierarchy levels. Panel (b) corresponds to the top left quarter of panel (a). . .	82
3.4	Overview of the one dimensional FMA implementation. Queries, keys, and values are aggregated hierarchically, and near field and far field interactions are evaluated without materializing dense low rank off diagonal blocks. . .	87
3.5	Construction of the bidirectional FMA operator with four hierarchy levels. The multilevel representations in panel (a) produce the sparse score blocks in panel (b). Panel (c) shows their conceptual superposition as the final hierarchical attention pattern; the implementation applies these blocks without materializing the full matrix.	89

3.6	Attention sparsity across three stages of a Swin Transformer. Different shades denote different stages; when cells overlap, the lightest shade, corresponding to the finest resolution, is shown. Unshifted layers attend within fixed local windows, while shifted layers allow cross window exchange along the displaced window boundaries.	90
3.7	Sample FMA attention maps from different layers and heads of the autoregressive enwik8 model with context length 1024. The empty upper triangles result from the causal mask.	95
3.8	Sample predictions from FMA2D-B on the ImageNet-1K validation set at 224×224 . Green labels indicate correct predictions with confidence scores. Red labels show misclassifications with the predicted and true classes. . . .	100
3.9	COCO 2017 validation AP during training. Left: when trained from scratch, FMA2D leads full attention and widens the gap. Right: with a pretrained full attention backbone, full attention retains the advantage.	103
3.10	Efficiency comparison of attention variants across input lengths. Left: peak GPU memory footprint. Right: inference latency.	105
4.1	Overview of MOSAIC-SR. Stage 1 aligns numerical observations with symbolic expressions through contrastive pretraining, and Stage 2 trains the decoder to predict operator structures with variable and constant placeholders. At inference time, MCTS maps these variable tokens to observed inputs, and the resulting equations initialize parallel search trajectories that alternate constant fitting with structural repair. The final expression is selected according to validation NMSE and complexity.	113
4.2	Predictive accuracy and symbolic solution rate on SRSD-Feynman without dummy variables. Points show mean performance on the Easy, Medium, and Hard splits. Horizontal bars show the reported variation across seeds. .	125
4.3	Rank based comparisons on SRSD-Feynman without dummy variables. The panels compare mean predictive accuracy rank with symbolic solution rank and expression complexity rank, respectively. Methods closer to the lower left corner achieve better tradeoffs.	126
4.4	Predictive accuracy and symbolic solution rate on SRSD-Feynman with dummy variables. Points show mean performance on the Easy, Medium, and Hard splits. Horizontal bars show the reported variation across seeds. .	127

4.5	Rank based comparisons on SRSD-Feynman with dummy variables. The panels compare mean predictive accuracy rank with symbolic solution rank and expression complexity rank, respectively. Methods closer to the lower left corner achieve better tradeoffs.	128
4.6	Best validation NMSE versus wall clock time on SRSD-Feynman Easy, grouped by ground truth operator count. The 150 runs comprise 30 equations evaluated over five seeds. The groups contain one or two, three, and four true operators, with 40, 55, and 55.	142
4.7	Best validation NMSE versus wall clock time on SRSD-Feynman Easy, grouped by top decoded sketch operator count. The runs are divided into sketches with fewer than 10, between 10 and 15, and more than 15 operators, with 57, 44, and 49 runs respectively.	143
4.8	Best validation NMSE versus wall clock time on SRSD-Feynman Medium, grouped by ground truth operator count. The 200 runs comprise 40 equations evaluated over five seeds. The groups contain fewer than five, five or six, and more than six true operators, with 75, 95, and 30 runs, respectively.	144
4.9	Best validation NMSE versus wall clock time on SRSD-Feynman Medium, grouped by top decoded sketch operator count. The groups contain fewer than 13, between 13 and 17, and more than 17 decoded operators, with 82, 53, and 65 runs respectively.	145
4.10	Best validation NMSE versus wall clock time during the 100 second search window on SRSD-Feynman Hard, grouped by ground truth operator count. The 250 runs comprise 50 equations evaluated over five seeds. The groups contain fewer than nine, between nine and 11, and more than 11 true operators, with 115, 75, and 60 runs, respectively.	146
4.11	Best validation NMSE versus wall clock time on SRSD-Feynman Hard, grouped by top decoded sketch operator count. The x axis spans 0–100 seconds. The 250 runs comprise 50 equations evaluated over five seeds. The groups contain fewer than 13, between 13 and 19, and more than 19 decoded operators, with 107, 73, and 70 runs, respectively.	147

List of Tables

3.1	Feature comparison of efficient attention mechanisms most relevant to FMA. Near field exact indicates whether exact attention is computed for nearby tokens or pixels. Far field aggregation describes how distant context is represented. Global per layer indicates whether each attention layer has a global receptive field. A dash indicates that the feature is not supported.	91
3.2	Autoregressive enwik8 results. Bold values are best among the subquadratic methods for a context length, and italics indicate second best. A multiplier in the memory column denotes the number of gradient accumulation passes.	94
3.3	Masked language modeling results on WikiText-103. Bold values are best among the subquadratic methods for a context length.	96
3.4	ImageNet-1K single crop classification without ImageNet-22K pretraining. Top 1 is measured in percent. Published P2T and CSWin values are included for context, and their memory is not directly comparable.	98
3.5	ImageNet-1K single crop classification with ImageNet-22K pretraining. Top 1 is measured in percent. Published CSWin values are included for context.	98
3.6	ADE20K validation results with ImageNet-22K pretraining. mIoU is measured in percent.	101
3.7	COCO 2017 validation results for Faster R-CNN. The ImageNet-1K initialization was trained using full attention before the attention operator was replaced.	102
3.8	Ablation on enwik8 with context length 1024. Each entry reports bpc followed by peak training memory. Avg pool replaces learned far field summaries by uniform average pooling.	103
3.9	End to end throughput on ImageNet-1K at 384×384	105

3.10	Best case FMA1D approximation of trained dense attention on WikiText-103.	106
3.11	Controlled symbol retrieval accuracy (%). Values are mean $\pm$ sample standard deviation across seeds 0, 1, 2. Nonadjacent averages the six depths from 0% to 90%. Local places the target immediately before the query. Far field attention remains enabled in the local control.	108
4.1	Core optimization settings used in the two pretraining stages.	120
4.2	Detailed Transformer architecture used by MOSAIC-SR, including the numeric encoder, symbolic encoder, and symbolic decoder. The complete architecture has approximately 17.4 million parameters and is relatively lightweight.	120
4.3	MOSAIC-SR inference time decoder, MCTS, and local-search budget for the cross-benchmark experiments.	121
4.4	SRSD-Feynman benchmark groups, variable count ranges, dummy variable distribution, data split sizes, and input domain extrema [103]. The extrema summarize equation specific sampling distributions rather than a shared interval. Dummy counts give the numbers of problems containing one, two, or three irrelevant variables.	122
4.5	Numbers of problems, variable count ranges, training and test sample counts for the Nguyen [156], Korns [74], Keijzer [67], Vladislavleva [165], Strogatz [21, 147], and Livermore [109] benchmarks.	122
4.6	Results on SRSD-Feynman. Easy, Medium, and Hard contain 30, 40, and 50 equations. Published baseline values are taken from the original SRSD-Feynman study. New evaluations performed for this chapter are means over five seeds. Acc is the percentage of equations with $R^2 > 0.999$, and Sym is the symbolic solution rate in percent. The top three values per column, including ties, are bolded. The best is also italicized.	123
4.7	Results on SRSD-Feynman with dummy variables. Published baseline values are taken from the original SRSD-Feynman study. New evaluations performed for this chapter are means over five seeds. Acc is the percentage of problems with test $R^2 > 0.999$, and Sym is symbolic solution rate in percent. The top three values in each column, including ties, are bolded. The best is also italicized.	124
4.8	Predictions for feynman-i.18.16.	129
4.9	Predictions for feynman-i.47.23.	129

4.10	Predictions for feynman-iii.15.27.	129
4.11	Predictions for feynman-ii.2.42.	130
4.12	Predictions for feynman-i.8.14.	130
4.13	Predictions for feynman-bonus.18.	131
4.14	Predictions for feynman-i.13.4.	131
4.15	Predictions for feynman-i.43.43.	131
4.16	Predictions for feynman-i.6.20.	132
4.17	Predictions for feynman-bonus.5.	132
4.18	Predictions for feynman-bonus.17.	133
4.19	Predictions for feynman-iii.21.20.	133
4.20	Results and runtime on six symbolic regression benchmarks. Sym and Acc are percentages. Time is the average runtime in seconds per equation. Acc denotes test NMSE below 10^{-6} . The top three Sym and Acc values per dataset are bolded.	134
4.21	Predictions for Nguyen-4.	135
4.22	Predictions for Nguyen-11.	135
4.23	Predictions for Strogatz-shearflow2.	136
4.24	Predictions for Livermore-1.	136
4.25	Predictions for Livermore-11.	137
4.26	Predictions for Livermore-21.	137
4.27	Taylor recovery for Logarithm.	138
4.28	Taylor recovery for Sine.	139
4.29	Taylor recovery for Feynman I.18.12.	139
4.30	Taylor recovery for Feynman I.12.11.	140
4.31	Component ablation on 30 clean SRSD-Feynman Easy equations. Acc and Sym report numerical and symbolic recovery in percent. All variants use seeds 0, 1, and 2.	148
4.32	Search budget ablation on 30 SRSD-Feynman Easy equations. Acc and Sym report numerical and symbolic recovery. All values are percentages. Paths are independent, and the columns are snapshots from continuous trajectories.149	

4.33	Recovery and planted sensor use on the 119 official Feynman datasets. Recovery is Match@5% on the test interior and extreme bins.	158
------	---	-----

List of Abbreviations

bpc Bits per Character 92

CvM Cramér-von Mises 68

FMA Fast Multipole Attention 1

FMM Fast Multipole Method 1

ICP Invariant Causal Prediction 65

IRM Invariant Risk Minimization 65

MCTS Monte Carlo Tree Search 62

mIoU mean Intersection over Union 93

PUCT Predictor + Upper Confidence bound applied to Trees 62

RIMS Residual Invariant Model Selection 152

SCM Structural Causal Model 64

SR Symbolic Regression 1

ViT Vision Transformer 3

List of Symbols

d Token embedding dimension 2

$\hat{d}$ Per-head query/key/value dimension in self attention 2

E Number of environments in causal symbolic regression 7

f^* True causal mechanism for the target variable 66

$\mathcal{G}_\Lambda$ Symbolic expression class induced by operator library Ω and complexity budget Λ 5

H, W Spatial height and width of a 2D feature map 4

n Sequence (or flattened image) length 2

X Observed covariates in symbolic regression 7

Y Target variable in symbolic regression 7

Chapter 1

Introduction

This thesis investigates two broad areas of machine learning: efficient Transformers and symbolic regression. The first part develops Fast Multipole Attention, a multilevel attention mechanism inspired by the Fast Multipole Method [42]. It preserves direct interactions between nearby tokens and represents distant interactions through learned summaries at multiple resolutions. A one dimensional construction is used for language modeling, and a two dimensional quadtree construction is used for image classification and semantic segmentation. The second part studies Transformer-guided symbolic regression for scientific equation recovery. MOSAIC-SR uses a pretrained Transformer to propose symbolic sketches that initialize tree search and parallel local searches for structural repair and constant fitting. RIMS is a related post hoc method that selects among candidate equations using residual invariance across environments. Both parts use structured representations to extend Transformers beyond standard dense sequence modeling: multilevel spatial structure supports efficient text and vision processing, while paired numerical and symbolic representations support scientific discovery.

These two projects share a focus on developing Transformer models and addressing several aspects of scientific machine learning. Fast Multipole Attention addresses computational efficiency through multilevel attention for text and vision. MOSAIC-SR addresses interpretability through closed form equation recovery, while the related RIMS selection method addresses robustness across environments. Before diving into in-depth background material in Chapter 2, this chapter presents a high-level overview of this thesis. Section 1.1 reviews the Transformer architecture and its main variants. Sections 1.2, 1.3, and 1.4 review the background for the two projects at a high level. Section 1.5 lists the objectives and contributions of the thesis. Section 1.6 gives the thesis overview.

1.1 The Transformer Architecture

The Transformer architecture [159] has become the dominant building block in modern deep learning. It was introduced as an encoder-decoder model for machine translation, replacing the recurrent processing of earlier sequence models [8] with attention between tokens. The encoder converts an input sequence into contextual representations. The decoder generates an output sequence one token at a time while attending to the encoder output. Each layer of the Transformer combines self attention with a positionwise feedforward network, residual connections, and layer normalization.

The architecture operates on embedded tokens rather than directly on raw data. Text is commonly divided into subword units. Byte Pair Encoding and SentencePiece keep frequent words compact while representing rare words through smaller reusable pieces [79, 137]. Data types other than text use different token definitions. An image is divided into patches. Audio can be represented by frames or learned latent units. A protein is represented by amino acid tokens. Once these objects are embedded as vectors, the same Transformer block can process them.

The central component is self attention. It takes a sequence of n tokens represented by a matrix $X \in \mathbb{R}^{n \times d}$ and uses learned linear projection matrices $W_Q, W_K, W_V \in \mathbb{R}^{d \times \hat{d}}$ for queries, keys and values. Here, $\hat{d}$ is the query, key, and value dimension of one attention head. The projections form

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \quad (1.1)$$

where $Q, K, V \in \mathbb{R}^{n \times \hat{d}}$. Self attention then produces

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{\hat{d}}}\right) V, \quad (1.2)$$

where the softmax is applied row by row to the $n \times n$ matrix $QK^\top$. The mechanism gives every token access to every other token in a single layer. This global receptive field is the source of much of the empirical strength of Transformers.

The queries, keys, and values are learned linear projections of the current token representations. A query describes the information sought by one token. A key determines how strongly another token matches that query. A value carries the information that is aggregated from that token position. Several attention heads are usually evaluated in parallel. Each head learns a separate set of projections, and their outputs are concatenated and

mixed by a learned linear map. This lets one layer represent several interaction patterns at the same time.

Positional information must be supplied separately because self attention alone does not distinguish token order. The original Transformer adds fixed sinusoidal encodings. Learned absolute embeddings and relative position representations are also common [138]. Relative schemes are useful when the relation between two positions matters more than their absolute indices. In images, position encodes spatial location. In music, speech, and biological sequences, it describes temporal or sequential distance.

Modern models often keep only part of the original design. Encoder-only models, such as BERT [35], use bidirectional self attention and are trained with masked token prediction. They produce contextual representations for classification, retrieval, and other understanding tasks. Decoder-only models, such as the GPT family [15], use causally masked self attention and are trained with next token prediction. They dominate text generation. The two families differ in their masking patterns and training objectives but share the same attention computation.

Encoder-decoder models retain both components. T5 expresses translation, summarization, classification, and question answering as text-to-text tasks [128]. BART pretrains an encoder-decoder by corrupting text and learning to reconstruct it [87]. These models are suited to conditional generation because the encoder represents the full input while the decoder produces an output under a causal mask.

Modern Transformers are usually pretrained on a broad dataset and then adapted to downstream tasks. BERT and RoBERTa learn bidirectional representations through masked language modeling [35, 97]. GPT-2 and GPT-3 learn through autoregressive next token prediction [15, 124]. Increasing model size, training data, and compute tends to improve language modeling performance in a predictable way [65]. Pretraining is therefore a major reason one architecture can support many tasks with limited task specific supervision.

Popular applications of Transformers include natural language understanding and generation, speech and music, biological sequence modeling, source code, scientific equation discovery, image classification, object detection, semantic segmentation, and video understanding. Text, audio, proteins, genomes, and code can be represented as one dimensional token sequences [6, 7, 39, 45, 58, 59, 61, 130], while unordered numerical observations can be encoded as sets for symbolic regression. In vision, the Vision Transformer (ViT) represents images as patch tokens [38]; DETR, Swin Transformer, Pyramid Vision Transformer, and SegFormer extend Transformer models to detection and dense prediction, and video models use spatiotemporal tokens [5, 19, 99, 171, 175].

Pretraining has also become central in vision. BEiT applies masked token prediction to image patches, and DINO learns visual representations by matching a student network to a teacher across different image views [9, 20]. For dense tasks, Deformable DETR attends to a small set of sampled image locations, and Mask2Former formulates several segmentation tasks through mask prediction [25, 191]. Segment Anything further shows how a large Transformer based vision model can produce masks from point or box prompts [68]. Transformers are also used for video understanding and image generation [12, 115].

Transformers also combine modalities. CLIP uses separate image and text encoders and aligns their representations through contrastive learning [127]. Fusion models instead allow tokens from different modalities to interact through self attention or cross attention [100]. Generative multimodal models condition a decoder on visual and textual inputs [2, 88]. These designs support retrieval, classification, question answering, and generation.

The broad applicability of Transformers comes from a common abstraction. Data become tokens, attention exchanges information, and pretraining learns reusable representations. This generality also makes attention cost important. Long documents, fine image grids, video clips, and multimodal inputs all increase token count. The first project modifies attention itself to make it scale better. The second uses a Transformer to predict symbolic structure from numerical data.

1.2 Efficient Self-Attention

The attention computation in Equation (1.2) forms an $n \times n$ matrix. The cost of dense self attention is therefore $\mathcal{O}(n^2\hat{d})$ in time and $\mathcal{O}(n^2)$ in memory. This grows quickly with sequence length. In long context language modeling, sequences of tens of thousands of tokens are common. In vision, an $H \times W$ feature map gives $n = HW$ tokens and the cost grows as $\mathcal{O}(H^2W^2)$, which is prohibitive even at moderate resolutions. The cost compounds in multimodal models that mix text and image at high resolution.

The quadratic cost affects many applications. Autoregressive language models need long context for generation over documents or conversations. Bidirectional encoders need longer inputs for document classification, question answering, and scientific sequence analysis [11, 186]. Vision backbones need fine spatial grids for image classification and dense prediction. Semantic segmentation is especially demanding because the output preserves spatial resolution [171, 175]. Multimodal systems process several input streams and often include many visual tokens in addition to text [100, 127]. These settings motivate attention mechanisms that preserve global communication without forming every pairwise interaction explicitly.

Attention efficiency is not a single objective. A useful method should reduce memory use and arithmetic cost, preserve accuracy, support the masking pattern required by the task, and remain practical on modern accelerators. These requirements can conflict. A local window is cheap but delays communication between distant tokens. A low rank approximation gives global coverage but can lose sharp local interactions. An implementation such as FlashAttention [32] computes exact dense attention more efficiently but retains quadratic arithmetic cost. The first part of this thesis focuses on the algorithmic question of how to preserve global communication with a multilevel computation.

A large literature aims to reduce this cost. Sparse and local attention restricts each query to a fixed subset of keys, often a sliding window or a structured pattern [11, 26, 99, 186]. Linearized attention replaces the softmax with a factorizable kernel [27, 66, 122]. Clustering and routing methods compute attention against learned centroids [69, 133]. Pooling and downsampling methods reduce the key and value sequence lengths through projection or pyramidal pooling [170, 171, 175]. Hierarchical methods organize the interaction across multiple resolutions and approach $\mathcal{O}(n)$ or $\mathcal{O}(n \log n)$ complexity [111, 187, 192]. Chapter 2 surveys these families in more detail.

The first project of this thesis, presented in Chapter 3, develops Fast Multipole Attention, which falls in the hierarchical family. It is motivated by the Fast Multipole Method [42] from n body simulation. In that setting, all pairs of particles interact but the interactions are computed in $\mathcal{O}(n)$ time by treating nearby particles directly at full resolution and summarizing distant particles by a compact representation. Fast Multipole Attention applies a similar decomposition to the attention matrix. Nearby tokens attend at full resolution. Distant tokens interact through learned summaries arranged in a multilevel hierarchy. The result is an attention mechanism with $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$ cost that still has a global receptive field in every layer.

1.3 Learning-Based Symbolic Regression

Symbolic regression is the task of recovering a closed form mathematical expression from observed data [29, 81]. A symbolic regression task is specified by a dataset $\{(x_i, y_i)\}_{i=1}^N$ with $x_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$, an operator library Ω , and a complexity budget Λ . The class of admissible expressions $\mathcal{G}_\Lambda$ is the set of expression trees built from Ω with complexity at most Λ . The class is finite for fixed Λ , but its size grows exponentially in the operator vocabulary and the budget, so the search is combinatorial.

A common formulation is to minimize the empirical squared loss with a complexity

penalty,

$$\hat{g} \in \arg \min_{g \in \mathcal{G}_\Lambda} \frac{1}{N} \sum_{i=1}^N (y_i - g(x_i))^2 + \lambda C(g), \quad (1.3)$$

where $C(g)$ is a complexity measure such as the number of nodes in the expression tree and $\lambda > 0$ controls the tradeoff. Two evaluation criteria are common. Numerical fit measures how close $g(x_i)$ is to y_i on a test set. Exact symbolic recovery requires that the returned g be algebraically equivalent to the ground truth equation after constant fitting. For example, $(x + 1)^2$ and $x^2 + 2x + 1$ have different expression trees but define the same function. The SRBench verifier parses and simplifies the predicted and ground truth expressions using SymPy, rounds fitted floating point constants, and examines their symbolic difference. It records a symbolic solution when the simplified difference is zero or constant, or when the ratio is constant [81]. This criterion recognizes algebraic rearrangements and expressions that differ only by an additive or multiplicative constant. Exact recovery is more demanding than numerical fit, because two expressions can fit the same data while differing structurally.

Classical symbolic regression systems use genetic programming [75]. Modern implementations such as PySR [29] and Operon [73] combine evolutionary search with Pareto front optimization on accuracy and complexity. These systems are strong baselines on standard benchmarks such as SRBench [81] and the AI Feynman dataset [154].

There are also more restricted forms of scientific equation discovery. SINDy [17] searches for a sparse governing equation inside a predefined dictionary of candidate terms. SISSO [113] uses compressed sensing to screen a large descriptor space. These approaches are effective when scientific knowledge provides a useful feature library. General symbolic regression considers a broader expression grammar and is therefore more flexible, but the larger search space makes exact recovery harder.

A more recent line of work treats symbolic regression as a supervised mapping from numerical data to symbolic expressions using neural networks. The Transformer architecture is the most common backbone. NeSymReS [13], end to end Transformer regression [63], and SNIP [141] train a Transformer encoder over the numerical data and a decoder that emits expression tokens. Inference is fast once the model is trained. The weakness is that exact recovery on hard benchmarks remains below that of classical search based systems, and gains have come more from hybridization with search than from larger end to end models. Monte Carlo tree search has been used to bring back search guided by a neural policy, in works such as TPSR [139].

This learning problem has a multimodal aspect. A mathematical function can be represented by a symbolic expression or by numerical input-output observations sampled

from the function. These two views contain related information but have different structure. The symbolic view is a token sequence or expression tree. The numerical view is a set of sampled points. SNIP aligns these views by contrastive pretraining [141]. Similar alignment objectives are widely used in image-text models such as CLIP [127]. Chapter 2 reviews this connection and the general pretrain and finetune pattern in more detail.

The second project of this thesis, presented in Chapter 4, follows the hybridization route. Instead of asking the Transformer to decode an expression token by token, the Transformer predicts structural information about the target expression, including the operator usage, the expression length, typed sketches, and motif probabilities. These predictions guide a Monte Carlo tree search procedure that performs the exact symbolic recovery through symbolic edits, numerical constant fitting, and algebraic verification. The neural model proposes where to search. The search performs the recovery. The method is evaluated on SRSD-Feynman with and without dummy variables and on six additional benchmarks: Nguyen-12, Korns-15, Keijzer-15, Vladislavleva-8, Strogatz-14, and Livermore-22.

1.4 Invariance in Symbolic Regression

Standard symbolic regression typically chooses a final expression by minimizing an objective that combines complexity with empirical risk computed after pooling observations from all environments. A causal setting assumes that the observed variables are generated by a directed system of mechanisms, so the direction of dependence matters [114, 118]. The causal parents of a target Y are the variables that directly determine Y , whereas a descendant is a variable generated downstream of Y , possibly through intermediate variables [118, 146]. For example, in the causal chain $X \rightarrow Y \rightarrow Z$, X is a parent of Y and Z is a descendant of Y . Because Z carries information about Y , an expression that predicts Y from Z can achieve very low error on pooled observational data. However, it reverses the causal direction and does not recover the mechanism that generates Y from its causes; it may therefore fail when an intervention changes the downstream relationship between Y and Z [114].

Mathematically, consider a target Y and observed covariates X recorded across several environments $e \in \{1, \dots, E\}$. An environment is a setting in which the data generating process is modified by an intervention on some subset of variables. For a candidate expression g , the residuals in environment e are

$$r_i^e(g) = y_i^e - g(x_i^e). \quad (1.4)$$

If g is the true mechanism that generates Y , the residuals reduce to the exogenous noise and follow the same distribution in every environment where the mechanism is unchanged. If g uses a descendant of Y or a confounded proxy, the residual distribution typically shifts across environments because the relationship between the surrogate and Y depends on the intervention.

The observation that the true mechanism produces invariant residuals is the basis of a line of work in causal inference. Invariant Causal Prediction [117] uses this principle to identify the causal parents of Y by testing whether $P(Y \mid X_S)$ is invariant across environments. Invariant Risk Minimization [4] and follow up methods such as anchor regression [132] apply related principles to representation learning. These methods produce variable sets, neural representations, or linear predictors, not symbolic equations.

The use of environments changes the model selection problem. A pooled score asks whether an expression predicts well on average. An invariance score asks whether the residual behavior remains stable when the surrounding system changes. This does not mean that every stable expression is causal. Stability can be uninformative if the available environments do not perturb the relevant parts of the system. Residual invariance therefore does not establish causality by itself. Instead, it helps distinguish a stable mechanism from a noncausal shortcut when the available interventions cause them to behave differently.

Next, we adapt the invariance principle to symbolic regression as a model selection rule. Across environments that preserve the target mechanism, the true equation leaves only the exogenous noise in its residuals and should therefore produce a stable residual distribution. A descendant based or otherwise noncausal expression may fit the pooled data well, but its residual distribution typically changes when the relationship between the shortcut variable and Y changes across environments. Thus, given a finite pool of candidate equations produced by a symbolic regression backend, the proposed selector ranks the candidates by the stability of their residual distributions across environments. A fit first, invariance second selection rule retains the candidates that are competitive on pooled fit and then prefers the most invariant among them. The selector is post hoc. It requires no retraining and can be applied to candidate pools generated by different symbolic regression systems. The second part of Chapter 4 establishes a finite-candidate recovery guarantee, describes the identification boundary, and evaluates RIMS on the official AI Feynman datasets with a planted measurement shortcut.

1.5 Objectives and Contributions

The objectives of this thesis are listed below.

- Reduce the cost of self attention while preserving a global receptive field.** The first contribution is Fast Multipole Attention, a multilevel attention mechanism for text and images. It uses direct local interactions at full resolution and learned far field summaries arranged in a multilevel hierarchy. The construction reduces the cost of dense $\mathcal{O}(n^2)$ self attention to $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$. A one dimensional variant is applied to autoregressive and bidirectional language modeling, where it achieves the best accuracy among the evaluated efficient attention baselines on enwik8 and WikiText-103 data. At a context length of 2048 on WikiText-103, FMA obtains a perplexity of 8.95, close to the 8.70 of quadratic full attention. A two dimensional variant with a quadtree partition is applied to image classification and semantic segmentation. FMA2D-L achieves 87.5% top-1 accuracy on ImageNet-1K and 52.4 mIoU on ADE20K, exceeding SegFormer-B5 by 0.9 mIoU points.
- Combine pretrained symbolic priors with explicit search for exact equation recovery.** The second contribution is MOSAIC-SR, a hybrid symbolic regression framework designed to close the gap between predictive fit and algebraic recovery. A recovery oriented Transformer is pretrained on normalized equations using algebraically equivalent rewrites as positive pairs and structurally similar but incorrect expressions as hard negatives. At inference, it proposes multiple symbolic sketches from a small subsample of numerical observations. Conditional Monte Carlo tree search maps decoded variable tokens to observed inputs and completes the sketches, after which parallel local searches alternate tree based structural repair with scale aware constant refitting. On SRSD-Feynman with and without dummy variables and six additional benchmarks, MOSAIC-SR achieves the highest symbolic solution rate in every reported dataset while ranking among the top two methods in predictive fit accuracy. Across the six cross-benchmark datasets, its improvement over the strongest baseline ranges from 12.5 to 33.3 percentage points, and it is 1.3–2.2 $\times$ faster than PySR.
- Select among candidate symbolic equations by residual invariance across environments.** The third contribution is a post hoc selector for causal symbolic regression. The selector ranks candidate equations by the stability of their residual distributions across environments, using a fit first, invariance second rule. It requires no retraining and applies to candidate pools produced by different symbolic regression backends. The chapter establishes a finite-candidate mechanism-class recovery guarantee under uniform estimation error and a detectable invariance gap, and shows that invariant downstream channels cannot be identified by residual invariance alone. On the 119 official AI Feynman datasets with a planted measurement sensor, RIMS

recovers 73 mechanisms at $\epsilon = 2.5$, compared with none under pooled fit; the physical mechanism is available in the union candidate pool for 81 datasets.

These contributions address three key requirements of scientific machine learning: efficiency, interpretability, and robustness. Fast Multipole Attention improves the efficiency of Transformers, MOSAIC-SR recovers interpretable closed form equations, and RIMS improves robustness by selecting equations that remain stable across environments.

1.6 Thesis Overview

The remainder of the thesis is organized as follows.

Chapter 2. Reviews technical background for the two projects, covering self attention and efficient attention variants, hierarchical and multilevel methods from numerical analysis, symbolic regression and its main solution families, and invariance principles in causal inference.

Chapter 3. Presents Fast Multipole Attention. Includes the one dimensional construction for sequences and the two dimensional quadtree construction for images, the complexity analysis, and experiments on autoregressive and bidirectional language modeling, image classification, semantic segmentation, and object detection. The experiments further show that FMA closely approximates learned full attention patterns and recovers most of the original model performance when used as a drop-in replacement for full attention. This chapter was written with Giang Tran and Hans De Sterck.

Chapter 4. Presents the symbolic regression project. The first part presents MOSAIC-SR, covering recovery oriented pretraining, sketch generation, conditional Monte Carlo tree search, parallel structural repair, scale aware constant fitting, and evaluations on SRSD-Feynman and six additional benchmarks. The second part presents RIMS, the residual invariance selector for causal symbolic regression, including the selection rule, finite-candidate guarantee, identification boundary, and evaluation on the official AI Feynman datasets with a planted measurement shortcut. This chapter is written with Peiyi Zheng, Giang Tran, and Hans De Sterck.

Chapter 5. Summarizes contributions, discusses limitations, and outlines future directions.

The chapters can also be read by topic. Readers interested in efficient Transformer architectures can read Chapters 2 and 3, with emphasis on Sections 2.1, 2.2, and 2.3.

Readers interested in symbolic regression can read Section 2.4 followed by Chapter 4. Section 4.2 additionally uses the causal inference background in Section 2.5.

Chapter 2

Background and Preliminaries

This chapter reviews the technical background for the two projects in the thesis. Section 2.1 surveys the Transformer architecture, training patterns, and applications, while Section 2.2 surveys the cost of dense attention and the broad landscape of efficient Transformer and attention variants. Section 2.3 reviews hierarchical numerical ideas that motivate Fast Multipole Attention, including the Fast Multipole Method, hierarchical matrices, and geometric tree partitions. Together, these sections provide the background for the Fast Multipole Attention project of Chapter 3. Section 2.4 surveys symbolic regression, with emphasis on classical search based systems, recent Transformer and large language model based systems, and Monte Carlo tree search. Section 2.5 reviews invariance and causal inference, including structural causal models, invariant prediction, and two sample tests for distributional discrepancy. These two sections provide the background for the symbolic regression project of Chapter 4, which includes MOSAIC-SR and RIMS.

Sections within this chapter are largely self contained. A reader interested in only one of the two projects can read the corresponding sections. These sections introduce different bodies of literature and use somewhat different notation. Notation conventions for each topic are stated at the start of the relevant section.

2.1 Transformers and Applications

This section reviews the Transformer architecture, its main training patterns, and its applications to one dimensional sequences, images, videos, and multimodal data. Material here is background for the Fast Multipole Attention construction presented in Chapter 3. The discussion of multimodal learning is also relevant to Chapter 4.

2.1.1 Transformer Architecture for One Dimensional Sequences

The Transformer architecture [159] was introduced for sequence to sequence learning. It replaced the recurrence used in earlier neural translation models [8] with stacks of attention and feedforward sublayers. This allows all token positions to be processed in parallel during training. In the original architecture, the *encoder* converts an input sequence into contextual representations, and the *decoder* generates an output sequence while attending to those encoder representations. Many later models keep only the encoder [35] or only the decoder [15, 124].

A Transformer does not operate on raw text. The input is first split into a sequence of discrete tokens by a tokenizer. In natural language, tokens may be words, characters, or subword units. Word-level tokenization keeps sequences short but needs a large vocabulary and handles rare or unseen words poorly. Character-level tokenization avoids unknown words but produces long sequences. Subword tokenization is the standard compromise. Byte Pair Encoding builds a vocabulary by repeatedly merging the most frequent adjacent symbol pair [137]. WordPiece [173] and SentencePiece [79] choose subword vocabularies by related but different criteria. A frequent word stays a single token, while a rare word such as **unhappiness** may be split into pieces such as **un**, **happi**, and **ness**. Vocabulary size sets a tradeoff. A larger vocabulary shortens sequences, and a smaller vocabulary shares more pieces across rare forms. For images, audio, and other data types, tokens need not be words or subwords. They can be image patches, acoustic frames, or learned discrete units, as discussed in Sections 2.1.4 and 2.1.5.

Each token index is mapped to a learned vector in $\mathbb{R}^d$. Let $E \in \mathbb{R}^{|\mathcal{V}| \times d}$ be an embedding table for a vocabulary $\mathcal{V}$. For token indices $t_1, \dots, t_n$ in a sequence of length n , the token embedding matrix is

$$X_{\text{tok}} = \begin{bmatrix} E_{t_1} \\ \vdots \\ E_{t_n} \end{bmatrix} \in \mathbb{R}^{n \times d}. \quad (2.1)$$

The row E_{t_i} is the vector assigned to the i th token index t_i . Thus $E_{t_1}, \dots, E_{t_n}$ are the learned token embeddings for the input sequence.

Token order must be added explicitly because attention alone is permutation equivariant: a reordering of the input rows produces the same reordering of the output rows and changes nothing else. A position matrix $P \in \mathbb{R}^{n \times d}$ is therefore added before the first layer,

$$H^{(0)} = X_{\text{tok}} + P. \quad (2.2)$$

Here X_{tok} represents token identities, P represents positions, and $H^{(0)}$ is the first hidden matrix passed to the Transformer blocks. Segment or modality embeddings can also be added when one sequence contains several input types. Figure 2.1 summarizes this input construction.

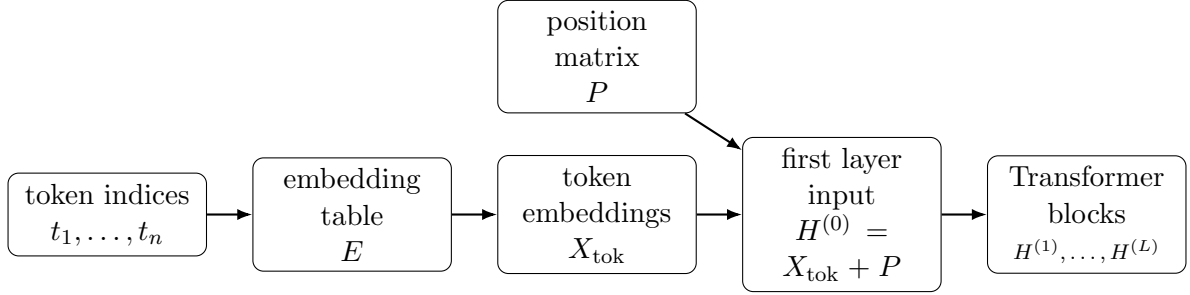


Figure 2.1: Input construction for a Transformer. Token indices are mapped to rows of the embedding table. Position information is added to form the first hidden matrix.

The original Transformer uses a fixed sinusoidal position matrix P . For position $\text{pos} \in \{0, \dots, n-1\}$ and embedding dimension-pair index $i \in \{0, \dots, \lfloor d/2 \rfloor - 1\}$, its entries are

$$\begin{aligned} P_{\text{pos}, 2i} &= \sin(\text{pos}/10000^{2i/d}), \\ P_{\text{pos}, 2i+1} &= \cos(\text{pos}/10000^{2i/d}). \end{aligned} \quad (2.3)$$

Each pair of embedding dimensions uses sine and cosine functions with the same wavelength. The wavelength varies across pairs. The resulting matrix P is deterministic and introduces no learned positional parameters.

Some models instead make attention depend on how far apart two tokens are. Relative schemes encode the displacement between two positions rather than the absolute index. The original relative position representation adds learned displacement dependent vectors to key and value interactions [138]. Rotary Position Embedding applies a position dependent rotation to queries and keys, so that their dot product depends on relative position [148]. ALiBi adds a head specific linear penalty to attention scores as distance grows [121]. These methods matter for long context models, because a model may be evaluated on sequences longer than those seen during pretraining.

A single attention head learns three projection matrices $W_Q, W_K, W_V \in \mathbb{R}^{d \times \hat{d}}$. Let $H \in \mathbb{R}^{n \times d}$ be the hidden matrix entering a layer. For the first layer, $H = H^{(0)}$. In *self attention*, the same matrix H supplies the queries, keys, and values,

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V. \quad (2.4)$$

The scaled dot product attention operation is

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{\hat{d}}}\right) V, \quad (2.5)$$

where the softmax is applied row by row to the matrix $QK^\top \in \mathbb{R}^{n \times n}$. Entry (i, j) of this matrix measures the compatibility between query i and key j . After the softmax, row i contains nonnegative weights that sum to one. The result of the attention operation is an $n \times \hat{d}$ matrix. Its i th row is the new representation for token i before the output projection, and it is a weighted average of all value vectors. Every token representation can therefore incorporate information from every input token in a single layer.

Some tasks need to remove part of this all pairs interaction pattern. Attention masks express which pairs of positions are allowed to interact. More generally, for one attention head, let $Q \in \mathbb{R}^{n_q \times \hat{d}}$ contain n_q queries and let $K, V \in \mathbb{R}^{n_k \times \hat{d}}$ contain n_k keys and values. A mask $M \in \{0, -\infty\}^{n_q \times n_k}$ has the same shape as the score matrix $QK^\top$. The masked operation is

$$\text{Attn}_M(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{\hat{d}}} + M\right) V. \quad (2.6)$$

Bidirectional self attention sets all entries of M to zero, apart from padding positions. Causal self attention sets $M_{ij} = -\infty$ when $j > i$. Local and sparse methods use the same expression with additional entries removed. Thus, the attention formula remains unchanged, while the mask specifies which query and key positions are allowed to interact.

A Transformer layer uses *multihead attention*: it evaluates several single-head attention operations in parallel rather than relying on one set of projections. For h heads, each head has separate learned projections,

$$\text{head}_r = \text{Attn}(HW_Q^{(r)}, HW_K^{(r)}, HW_V^{(r)}), \quad r = 1, \dots, h. \quad (2.7)$$

The head outputs are concatenated and mixed by a learned output projection $W_O \in \mathbb{R}^{h\hat{d} \times d}$,

$$\text{MHA}(H) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_O. \quad (2.8)$$

Different heads can represent different interaction patterns. Some heads concentrate on local context, while others can carry information over longer distances. The matrix W_O mixes the h separate value mixtures back into one d dimensional representation per token. Thus the query, key, and value projections occur before each attention head, whereas the output projection occurs after the head outputs are concatenated.

Self attention uses the same sequence to form queries, keys, and values. *Cross attention* uses two sequences. If H_D is a decoder representation and H_E is an encoder representation, then the queries, keys, and values in cross attention are given by

$$Q = H_D W_Q, \quad K = H_E W_K, \quad V = H_E W_V. \quad (2.9)$$

The decoder can then retrieve different parts of the encoded input at each output position. Cross attention also appears in multimodal models. With more than two modalities, a model can concatenate the key–value sequences from several modalities, apply separate cross-attention modules to them, or first fuse all modality tokens with self attention; the architectural choice determines which representations can interact directly.

The encoder in the original Transformer [159] is a stack of bidirectional self-attention layers, so every input token can attend to every input position. The original decoder contains masked self attention followed by encoder–decoder cross attention [159]. In masked self attention, token i can attend only to tokens at positions $1, \dots, i$, which prevents the decoder from using future output tokens. Cross attention then uses decoder queries and encoder keys and values, as in Equation (2.9).

The second part of a Transformer layer is a positionwise feedforward network. It is applied independently to each token,

$$\text{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2, \quad (2.10)$$

where $W_1 \in \mathbb{R}^{d \times d_{\text{ff}}}$ and $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$ in row vector notation. That notation treats x as a $1 \times d$ row, so xW_1 has size $1 \times d_{\text{ff}}$ and the final product with W_2 returns to size $1 \times d$. The intermediate width d_{ff} is usually several times larger than the model width d . The activation σ is applied componentwise. Early Transformers used ReLU. Later language and vision models often use smoother or gated variants, such as GELU or gated linear units. For a scalar z ,

$$\text{ReLU}(z) = \max\{z, 0\}, \quad \text{GELU}(z) = z\Phi(z) = \frac{z}{2} \left(1 + \text{erf}\left(\frac{z}{\sqrt{2}}\right) \right), \quad (2.11)$$

where Φ is the standard normal cumulative distribution function. ReLU and GELU are applied componentwise to vectors. A gated linear unit uses two affine projections of the same input and multiplies one projection by a learned gate,

$$\text{GLU}(x) = (xW_a + b_a) \odot \text{sigmoid}(xW_b + b_b), \quad \text{sigmoid}(z) = \frac{1}{1 + e^{-z}}. \quad (2.12)$$

In a gated feedforward block, this gated hidden vector is then projected back to the model width. The feedforward network does not mix information across positions. It mixes

channels within each token vector after attention has already exchanged information across the sequence.

Layer normalization is another basic component of a Transformer block. It normalizes each token vector independently across its feature channels. For $x \in \mathbb{R}^d$, define

$$\mu(x) = \frac{1}{d} \sum_{r=1}^d x_r, \quad s^2(x) = \frac{1}{d} \sum_{r=1}^d (x_r - \mu(x))^2. \quad (2.13)$$

LayerNorm then has the form

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu(x)\mathbf{1}}{\sqrt{s^2(x) + \epsilon}} + \beta, \quad (2.14)$$

where $\gamma, \beta \in \mathbb{R}^d$ are learned scale and shift parameter vectors, $\epsilon > 0$ is a small numerical constant, and $\odot$ denotes componentwise multiplication. Since normalization is performed within each token, LayerNorm does not depend on batch statistics. This is important for sequence models, where batch size, sequence length, padding, and generation mode may vary.

Residual connections are applied around the attention and feedforward sublayers in a block. Ignoring the batch dimension, let $H \in \mathbb{R}^{n \times d}$ be the sequence representation entering a sublayer and let $F : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times d}$ denote the sublayer transformation, including any output projection needed to preserve the model dimension. The residual update is

$$H^+ = H + F(H), \quad H, F(H), H^+ \in \mathbb{R}^{n \times d}. \quad (2.15)$$

For a batch of b sequences, the same operation is applied to tensors of shape $b \times n \times d$. It adds the sublayer output back to its input and therefore preserves a direct path for information and gradients. Without this path, very deep stacks of attention and feedforward layers are harder to optimize. The residual path also lets a layer make a small correction to the current representation rather than replacing it completely. This is the residual learning view introduced by He et al. [51].

There are two common normalization orders. The original Transformer uses post norm [159]. In the formulas below, Sublayer denotes the transformation inside the residual branch. It can be a multihead self attention block, a positionwise feedforward block, or a cross attention block in a decoder. A generic sublayer updates its input by

$$x^+ = \text{LayerNorm}(x + \text{Sublayer}(x)). \quad (2.16)$$

Many modern implementations use pre norm instead [176],

$$x^+ = x + \text{Sublayer}(\text{LayerNorm}(x)). \quad (2.17)$$

Both forms combine the same operations. The difference is where the normalized signal enters the sublayer and where the residual path sits. Pre norm gives a more direct identity path through the stack, which is helpful when the number of layers is large. Post norm can make the representation after each sublayer well scaled, but the residual path passes through LayerNorm at every layer.

The attention and feedforward sublayers have complementary roles. Attention mixes information across token positions. The feedforward network mixes channels independently at each position and usually expands the hidden dimension before projecting it back to d . In a common pre norm layer, the updates are

$$\begin{aligned} \tilde{H}^{(\ell)} &= H^{(\ell)} + \text{MHA}(\text{LayerNorm}(H^{(\ell)})), \\ H^{(\ell+1)} &= \tilde{H}^{(\ell)} + \text{FFN}(\text{LayerNorm}(\tilde{H}^{(\ell)})). \end{aligned} \quad (2.18)$$

Some decoder implementations also apply a final LayerNorm after the last block before the prediction head. This keeps the scale of the final hidden states controlled even when many residual updates have been accumulated.

Figure 2.2 summarizes where the projections, attention modules, residual paths, and feedforward networks occur. The diagram shows the post-norm order of the original Transformer; a pre-norm block moves each LayerNorm to the input of the corresponding attention or feedforward sublayer, as in Equation (2.18).

Dropout is commonly applied during training. It may be applied to token and position embeddings, to attention probabilities after the softmax, or to the outputs of attention and feedforward sublayers before the residual addition. Attention dropout randomly removes some pairwise interactions during training. Residual dropout randomly removes part of a sublayer update. Both act as regularizers. At inference time dropout is disabled, so the block is deterministic once the model parameters and decoding rule are fixed.

Three common model families arise from this design. Encoder-only models, such as BERT [35], produce contextual representations for classification, extraction, and retrieval. Decoder-only models, such as the GPT family [15, 124], generate sequences autoregressively. Encoder-decoder models, such as T5 [128], map an input sequence to an output sequence. Translation, summarization, and structured generation naturally fit the encoder-decoder setting. The same building blocks can also process non-text inputs after those inputs have been converted into tokens.

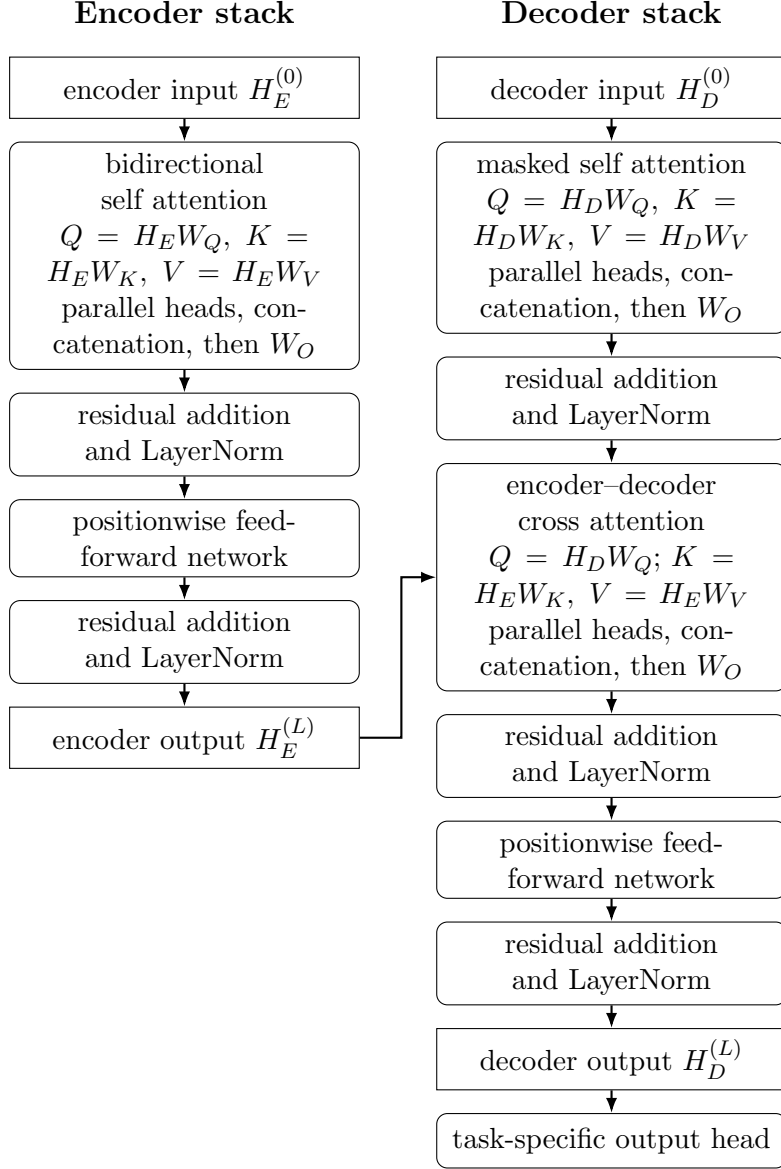


Figure 2.2: Encoder–decoder Transformer data flow as in the original Transformer [159]. For simplicity, only one Transformer block is represented in each of the Encoder and Decoder. Each attention module contains query, key, and value projections before its heads and an output projection after the head outputs are concatenated. Encoder-only models retain the encoder stack, decoder-only models retain the masked decoder stack without encoder cross attention, and encoder–decoder models use both stacks.

The final representation is connected to a task-specific output map. Write the rows of the last hidden matrix as $H^{(L)} = [h_1; \dots; h_n]$. For C output classes, a token-classification head applies the same linear map to every row, whereas a sequence-classification head first forms a pooled representation h_{pool} , for example from a summary token or mean pooling:

$$\begin{aligned} p_{\text{tok}}(y_i | x) &= \text{softmax}(h_i W_{\text{tok}} + b_{\text{tok}}), & W_{\text{tok}} &\in \mathbb{R}^{d \times C}, \\ p_{\text{seq}}(y | x) &= \text{softmax}(h_{\text{pool}} W_{\text{seq}} + b_{\text{seq}}), & W_{\text{seq}} &\in \mathbb{R}^{d \times C}. \end{aligned} \quad (2.19)$$

A language-model head instead maps each decoder state to a distribution over the vocabulary,

$$p_{\theta}(x_{i+1} | x_{\leq i}) = \text{softmax}(h_i W_{\text{vocab}} + b_{\text{vocab}}). \quad (2.20)$$

Here $W_{\text{vocab}} \in \mathbb{R}^{d \times |\mathcal{V}|}$ is often tied to the transpose of the input embedding table, $W_{\text{vocab}} = E^{\top}$.

Training and autoregressive inference use the same parameters but different execution patterns. Under a causal mask, training evaluates predictions for all positions in parallel because the target prefix is known. During generation, the model produces one new token at a time. Keys and values from earlier positions are cached so they do not need to be recomputed. For one layer with h heads, this key-value cache stores $2nh\hat{d}$ numbers and therefore uses $\Theta(nh\hat{d})$ memory, or $\Theta(nd)$ when $h\hat{d} = d$. Across L layers it uses $\Theta(Lnd)$ memory under the same convention. Each new query must still interact with all cached keys under dense attention, which creates a latency bottleneck even when only one token is generated at a time.

For hidden width d and sequence length n , dense attention has a token interaction cost of $\mathcal{O}(n^2d)$. During ordinary training, the attention scores for h heads form an $h \times n \times n$ tensor, so materializing them requires $\Theta(hn^2)$ memory. The query, key, value, output, and feedforward projections add terms that scale linearly with n and quadratically with width. At short sequence lengths, these dense channel projections can dominate. At long sequence lengths, the n^2 interaction and memory terms become dominant. Efficient attention methods target this latter regime.

2.1.2 Vision Transformer Architecture and Visual Backbones

Images are two dimensional arrays rather than one dimensional token sequences. This subsection describes the Vision Transformer in detail, including patch tokenization, position information, the encoder backbone, and hierarchical extensions that preserve multiscale spatial structure.

To convert a two dimensional image into Transformer input, a simple approach is to divide it into nonoverlapping patches and treat each patch as a token. For an image of height H , width W , and patch side length p , the number of tokens is

$$n = \frac{HW}{p^2}. \quad (2.21)$$

Let $I_1, \dots, I_n \in \mathbb{R}^{p \times p \times c}$ be patches from an image with c channels. A linear projection $E_p \in \mathbb{R}^{p^2 c \times d}$ maps each flattened patch to model width d . With a learned class token x_{cls} and position embeddings P , the input is

$$X^{(0)} = [x_{\text{cls}}; \text{vec}(I_1)E_p; \dots; \text{vec}(I_n)E_p] + P. \quad (2.22)$$

Here $x_{\text{cls}} \in \mathbb{R}^d$ and $P \in \mathbb{R}^{(n+1) \times d}$. The sequence has $n+1$ rows: row zero is the class token and rows $1, \dots, n$ correspond to image patches. The same encoder parameters are applied to all rows, so the patch projection determines the spatial granularity while the class token supplies a dedicated global readout.

Vision Transformer (ViT) [38] applies L pre-normalization Transformer blocks to this sequence. For block $\ell = 1, \dots, L$,

$$\begin{aligned} U^{(\ell)} &= X^{(\ell-1)} + \text{MSA}(\text{LN}(X^{(\ell-1)})), \\ X^{(\ell)} &= U^{(\ell)} + \text{MLP}(\text{LN}(U^{(\ell)})), \\ \text{MLP}(Z) &= \text{GELU}(ZW_1 + b_1)W_2 + b_2, \end{aligned} \quad (2.23)$$

where $W_1 \in \mathbb{R}^{d \times d_{\text{ff}}}$ and $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$. Multihead self attention (MSA) is the operation in Equation (2.8), now applied to the class token and all patch tokens together. Consequently, every layer can exchange information between arbitrary image regions. For classification, the final class-token row is normalized and mapped to class probabilities,

$$p_{\theta}(y \mid I) = \text{softmax}\left(\text{LN}\left(X_{0,:}^{(L)}\right) W_{\text{cls}} + b_{\text{cls}}\right). \quad (2.24)$$

Mean pooling over $X_{1:n,:}^{(L)}$ is an alternative readout. Thus Equation (2.22) is the visual counterpart of the text input construction in Equations (2.1)–(2.2).

This construction is called *visual tokenization*: it converts image regions into the vectors consumed by a Transformer. It differs from text tokenization. A text tokenizer usually maps each subword to a discrete vocabulary index before the model begins. Standard image patchification does not assign a discrete identity to each patch. It flattens the $p \times p \times c$ pixel array and applies the learned matrix E_p in Equation (2.22), producing a real-valued

vector in $\mathbb{R}^d$. The same projection can be implemented as a convolution with kernel size and stride both equal to p ; because the stride equals the kernel size, adjacent input regions do not overlap.

This nonoverlapping construction is *fixed patchification*. It is simple and preserves a direct correspondence between tokens and image regions. It also imposes hard patch boundaries. Pixels on opposite sides of a boundary do not interact before the first attention layer, even when they belong to the same edge or texture. Every image receives the same number of tokens, including images whose useful content is concentrated in only a small region. These properties make fixed patches easy to batch and index, but they provide little image specific inductive bias.

Convolutional tokenization adds local processing before global attention. Let $J_{i,j} \in \mathbb{R}^{k \times k \times c}$ be the image region extracted by a kernel of size k and stride s . An overlapping patch embedding has the form

$$x_{i,j} = \text{vec}(J_{i,j})E_k, \quad H' = \left\lfloor \frac{H-k}{s} \right\rfloor + 1, \quad W' = \left\lfloor \frac{W-k}{s} \right\rfloor + 1, \quad (2.25)$$

where $E_k \in \mathbb{R}^{k^2 c \times d}$ and the output contains $H'W'$ tokens when no padding is used. Fixed ViT patchification is the special case $k = s = p$. When $s < k$, adjacent tokens share pixels and the tokenizer supplies a local continuity bias.

CvT [172] uses convolutional token embeddings and convolutional projections inside attention, so queries, keys, and values are formed after local spatial mixing,

$$Q = \mathcal{C}_Q(X)W_Q, \quad K = \mathcal{C}_K(X)W_K, \quad V = \mathcal{C}_V(X)W_V, \quad (2.26)$$

where each $\mathcal{C}$ is a spatial convolution applied on the token grid. Tokens-to-Token ViT [184] instead repeatedly extracts overlapping neighborhoods, applies local attention, and restructures the resulting tokens into a shorter grid. Both methods move local aggregation into the early stages, where edges and textures can be represented before expensive global interaction.

Token reduction means replacing a fine token grid by a shorter representation after the initial tokens have been formed. Swin Transformer and Pyramid Vision Transformer reduce tokens according to a regular spatial hierarchy [99, 171]. TokenLearner instead constructs $m \ll n$ content-adaptive tokens from $X = [x_1; \dots; x_n]$ using learned spatial weighting functions $a_k : \mathbb{R}^d \rightarrow \mathbb{R}$,

$$\alpha_{ki} = \frac{\exp(a_k(x_i))}{\sum_{j=1}^n \exp(a_k(x_j))}, \quad z_k = \sum_{i=1}^n \alpha_{ki} x_i, \quad k = 1, \dots, m. \quad (2.27)$$

Each z_k summarizes the regions selected by one weighting map [134]. This lets the model allocate computation to informative content rather than to every location uniformly. The tradeoff is that the reduced tokens no longer have a fixed one-token-to-one-patch correspondence, which complicates dense pixel-level prediction.

Real-valued and discrete visual tokens serve different purposes. A real-valued token is a vector in $\mathbb{R}^d$, whereas a discrete visual token is an index into a finite learned codebook. ViT, CvT, and hierarchical vision Transformers use real-valued embeddings that are trained jointly with the backbone. BEiT [9] also uses real-valued patch embeddings as model input, but its masked prediction targets are discrete codebook indices produced by a separate image tokenizer. This finite vocabulary makes masked image modeling resemble masked language modeling, and prediction quality depends on what visual information the separate tokenizer preserves.

Patch size sets a direct information and computation tradeoff. Larger patches shorten the token sequence but combine many pixels before attention is applied. Fine boundaries and small objects can then be difficult to recover. Smaller patches preserve more detail, but Equation (2.21) shows that halving the patch side length increases the number of tokens by a factor of four. Dense attention cost then increases by a factor of sixteen. This tradeoff is mild for low resolution classification and is often severe for detection, segmentation, and video.

The tokenizer therefore determines more than input format. It controls sequence length, local receptive fields, spatial correspondence, and the amount of information discarded before attention. Classification can tolerate aggressive pooling because only a global output is needed. Detection and segmentation need finer tokens and explicit spatial geometry.

Position embeddings identify the row and column of each patch. Learned absolute embeddings are simple, but their size depends on the training resolution. When a model is applied at another resolution, the two dimensional position grid is commonly interpolated. Relative position biases instead modify the attention score using the row and column displacement between two patches,

$$A_{ij} = \frac{q_i^\top k_j}{\sqrt{\hat{d}}} + b_{\Delta r_{ij}, \Delta c_{ij}}, \quad (2.28)$$

where $(\Delta r_{ij}, \Delta c_{ij})$ is the relative offset from patch i to patch j . Sharing $b_{\Delta r, \Delta c}$ across all locations makes the bias depend on spatial displacement rather than absolute image coordinates. Unlike a convolutional network, a plain ViT has little built-in locality or translation bias. It must learn these patterns from data, which is why it depends more heavily on large scale pretraining.

Dense recognition tasks need feature maps at several spatial resolutions, in the way that convolutional backbones produce a pyramid. A hierarchical Transformer begins with fine spatial tokens and repeatedly merges neighboring patches. Spatial resolution decreases between stages while channel width and receptive field increase. Early stages retain edges, textures, and small structures; later stages represent larger parts and whole objects.

Let $Z^{(s)} \in \mathbb{R}^{H_s \times W_s \times d_s}$ be the feature grid at stage s . A standard 2×2 patch-merging operation concatenates four neighboring vectors and applies a learned projection,

$$\begin{aligned} m_{i,j}^{(s)} &= \left[z_{2i,2j}^{(s)}; z_{2i+1,2j}^{(s)}; z_{2i,2j+1}^{(s)}; z_{2i+1,2j+1}^{(s)} \right] \in \mathbb{R}^{4d_s}, \\ z_{i,j}^{(s+1)} &= \text{LN}\left(m_{i,j}^{(s)}\right) W_{\text{merge}}^{(s)}, \quad W_{\text{merge}}^{(s)} \in \mathbb{R}^{4d_s \times d_{s+1}}. \end{aligned} \quad (2.29)$$

The spatial size changes from $H_s \times W_s$ to $(H_s/2) \times (W_s/2)$, while the channel dimension usually increases. The number of tokens falls by a factor of four, so later blocks can process larger receptive fields at lower cost.

Swin Transformer [99] combines patch merging with attention inside nonoverlapping windows each containing $M \times M$ patches. With $n_s = H_s W_s$ tokens, the interaction costs of global and window attention are

$$\mathcal{C}_{\text{global}} = \mathcal{O}(n_s^2 d_s), \quad \mathcal{C}_{\text{window}} = \frac{n_s}{M^2} \mathcal{O}(M^4 d_s) = \mathcal{O}(n_s M^2 d_s). \quad (2.30)$$

For fixed M , window attention is linear in the number of image tokens. Alternating blocks shift the window grid by approximately half a window, so tokens separated by an earlier boundary can interact in the next block. Repeated shifting and patch merging gradually expand the receptive field without applying full attention at the finest resolution.

Pyramid Vision Transformer (PVT) also forms a multistage feature pyramid, but reduces the key and value grids inside attention [171]. For a spatial reduction ratio r_s , write

$$Q_s = X_s W_Q, \quad K_s = \text{SR}_{r_s}(X_s) W_K, \quad V_s = \text{SR}_{r_s}(X_s) W_V, \quad (2.31)$$

where $X_s \in \mathbb{R}^{n_s \times d_s}$ and $\text{SR}_{r_s}(X_s)$ contains approximately n_s/r_s^2 spatial tokens. The attention-score matrix therefore has size $n_s \times (n_s/r_s^2)$ rather than $n_s \times n_s$, reducing its interaction cost to $\mathcal{O}(n_s^2 d_s/r_s^2)$. Swin preserves full token resolution within local windows, whereas PVT lets each query attend globally to a reduced key–value representation. The detailed relationship of these mechanisms to other efficient attention methods is discussed in Section 2.2.

2.1.3 Pretraining and Downstream Adaptation

Modern Transformer models are usually trained in two stages. First, a large model is pre-trained on a broad data distribution with a self-supervised or weakly supervised objective. This stage learns reusable token representations, attention patterns, and prediction heads. Second, the pretrained model is adapted to one or more downstream tasks. Downstream adaptation is broader than finetuning. It may update all model weights, train only a small output head, learn a small set of extra parameters, or use only a changed input prompt. Pretraining is expensive, but the resulting representation can be reused. Adaptation then uses less task specific data and compute than training the same architecture from scratch.

Autoregressive language modeling trains a decoder to predict the next token. For a sequence $x_1, \dots, x_n$, write $x_{<i} = (x_1, \dots, x_{i-1})$. The objective is

$$\mathcal{L}_{\text{AR}} = - \sum_{i=1}^n \log p_{\theta}(x_i \mid x_{<i}). \quad (2.32)$$

For $i = 1$, the context $x_{<1}$ is empty or contains a designated beginning-of-sequence token. The conditional distribution p_{θ} is produced by the decoder and its vocabulary head, as in Equation (2.20). The causal mask ensures that prediction at position i cannot depend on future tokens. Each position supplies a supervised prediction target created from raw text itself, so no separate human label is needed for the basic pretraining objective. GPT-2 showed that a single decoder trained in this way can transfer across many language tasks [124]. GPT-3 showed that scaling the decoder, training corpus, and compute budget can give strong in context learning from instructions and examples placed in the context window [15].

Masked language modeling trains a bidirectional encoder. A random subset $\mathcal{M} \subseteq \{1, \dots, n\}$ of token positions is hidden, and the model predicts the hidden tokens from visible context on both sides,

$$\mathcal{L}_{\text{MLM}} = - \sum_{i \in \mathcal{M}} \log p_{\theta}(x_i \mid x_{\setminus \mathcal{M}}). \quad (2.33)$$

Here $x_{\setminus \mathcal{M}}$ denotes the input sequence with the positions in $\mathcal{M}$ removed or replaced by masking symbols. Thus the loss is summed only over the hidden target positions. BERT [35] showed that a pretrained bidirectional encoder can be adapted to question answering, natural language inference, and other language understanding tasks by adding a small output head and finetuning all parameters. RoBERTa [97] later showed that the training recipe, data scale, and batch size have large effects even when the architecture is nearly

unchanged. Encoder-decoder models use related denoising objectives. T5 [128] studies this transfer setting by expressing many language tasks in a common text-to-text form. BART [87] corrupts an input sequence and trains the model to reconstruct it, which suits generation tasks such as summarization and translation.

The objectives above describe pretraining. Once those reusable parameters have been learned, downstream adaptation changes the model or its input for a particular task. The most direct adaptation method is supervised finetuning. Given labeled data $\mathcal{D}_{\text{sup}} = \{(x_i, y_i)\}_{i=1}^m$, where x_i is the i th task input, y_i is its target, and m is the number of labeled examples, the usual negative log-likelihood loss is

$$\mathcal{L}_{\text{sup}}(\theta) = - \sum_{i=1}^m \log p_{\theta}(y_i | x_i). \quad (2.34)$$

For classification, the Transformer first summarizes the final hidden matrix as $h_{\text{pool}} \in \mathbb{R}^d$, using either a designated summary token or a pooling operation. This vector is connected to a task head,

$$p_{\theta}(y | x) = \text{softmax}(h_{\text{pool}}W + b), \quad (2.35)$$

where $W \in \mathbb{R}^{d \times C}$ and $b \in \mathbb{R}^C$ for C classes. Full finetuning updates the pretrained backbone together with this head. It usually gives high accuracy on the particular downstream task, but it stores gradients and optimizer states for all trainable parameters. Therefore, full finetuning requires substantially more memory and may force the use of smaller batches, shorter sequences, or smaller models. It can also overfit when the downstream dataset is small or narrow.

Linear probing freezes the pretrained backbone and trains only a simple output map. It is a common way to test whether the representation already separates the downstream classes. It is cheaper and more stable than full finetuning, but it cannot change the features produced by the attention layers. Partial finetuning sits between these extremes. It freezes most layers and updates only the last few blocks or the task head.

Prompting adapts a model by changing the input rather than the weights. Instructions, task descriptions, or a few examples are placed in the context window. A decoder then produces the answer with the same pretrained parameters. This is useful when training examples are unavailable or when one model must be used for many tasks without storing separate parameter sets. It also has a cost. The prompt consumes context length, and performance can depend strongly on wording, example order, and decoding rule.

Parameter efficient adaptation occupies a middle ground between full finetuning and prompting. Prefix tuning [92] keeps the language model fixed and optimizes task-specific

real-valued vectors that are not vocabulary tokens but act as additional learned context. If the prefix has m_p positions, then $K_{\text{prefix}}, V_{\text{prefix}} \in \mathbb{R}^{m_p \times \hat{d}}$. Later tokens can attend to these vectors. In attention notation, prefix tuning augments the keys and values,

$$K' = \begin{bmatrix} K_{\text{prefix}} \\ K \end{bmatrix}, \quad V' = \begin{bmatrix} V_{\text{prefix}} \\ V \end{bmatrix}. \quad (2.36)$$

A second parameter-efficient method, LoRA [57], freezes pretrained weights and learns low rank updates inside Transformer layers. For a linear map with weight $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ in row vector notation, the effective weight is

$$W_{\text{eff}} = W + AB, \quad A \in \mathbb{R}^{d_{\text{in}} \times r}, \quad B \in \mathbb{R}^{r \times d_{\text{out}}}, \quad (2.37)$$

where $r \ll \min(d_{\text{in}}, d_{\text{out}})$. Only the small matrices A and B are trained. These methods reduce the number of trainable parameters and make it practical to retain separate task modules for several downstream settings. They do not change the basic attention operator. Instead, they reduce adaptation-time memory and computation and reduce the storage needed for each task-specific model; these concerns are distinct from the sequence-length cost of attention during pretraining and inference.

Vision Pretraining. Vision backbones can be pretrained with class labels, discrete tokenizer targets, teacher predictions, or reconstructed pixels. These objectives supervise different parts of the representation: classification emphasizes global semantics, masked prediction emphasizes patch content and context, and reconstruction requires the encoder to preserve information useful for recovering local image structure.

Given a minibatch $\{(I_i, y_i)\}_{i=1}^B$ and a random image augmentation a , supervised ViT training minimizes the cross entropy of the class-token prediction in Equation (2.24),

$$\mathcal{L}_{\text{sup}} = -\frac{1}{B} \sum_{i=1}^B \log p_{\theta}(y_i \mid a(I_i)). \quad (2.38)$$

The augmentations change color, crop, scale, or other nuisance properties while preserving the target class, encouraging the encoder to learn invariances that transfer beyond the training images.

DeiT [152] adds a learned distillation token to the input sequence. Its final representation is passed to a separate distillation head that learns from a pretrained teacher. Let p_{cls} be the student class-head distribution, $p_{\text{dist}}^{(\tau)}$ the temperature-scaled distillation head, and $q_T^{(\tau)}$ the teacher distribution. A soft distillation objective can be written as

$$\mathcal{L}_{\text{DeiT}} = (1 - \lambda) \text{CE}(y, p_{\text{cls}}) + \lambda \tau^2 \text{KL}\left(q_T^{(\tau)} \parallel p_{\text{dist}}^{(\tau)}\right). \quad (2.39)$$

Here λ balances ground-truth and teacher supervision, and τ controls the softness of the class probabilities. Hard distillation replaces $q_T^{(\tau)}$ by the teacher’s predicted class. At inference, the class and distillation heads can be averaged. This additional supervision helped make ViT training effective on ImageNet-1K without a much larger private dataset [34].

BEiT [9] separates the input representation from the prediction target. The Transformer still receives real valued patch embeddings, but a pretrained image tokenizer assigns a discrete code $z_j \in \{1, \dots, K\}$ to each patch. For a masked index set $\mathcal{M}$, the encoder observes the corrupted image $I_{\setminus \mathcal{M}}$ and predicts the missing codes,

$$\mathcal{L}_{\text{BEiT}} = - \sum_{j \in \mathcal{M}} \log p_{\theta}(z_j \mid I_{\setminus \mathcal{M}}). \quad (2.40)$$

This is the visual analogue of masked language modeling in Equation (2.33): both objectives predict discrete identities from bidirectional context. The learned representation depends partly on the codebook, because the tokenizer determines which visual distinctions count as different targets.

DINO [20] does not require class labels or a discrete image tokenizer. A student and a teacher receive different augmented views $a_v(I)$ and $a_u(I)$ of the same image. Their output distributions are

$$p_s^{(v)} = \text{softmax}\left(\frac{g_{\theta}(a_v(I))}{\tau_s}\right), \quad p_t^{(u)} = \text{softmax}\left(\frac{g_{\xi}(a_u(I)) - c}{\tau_t}\right), \quad (2.41)$$

where g_{θ} and g_{ξ} are the student and teacher networks, c is a running center, and τ_s, τ_t are temperatures. The student matches the teacher across distinct views,

$$\mathcal{L}_{\text{DINO}} = - \sum_{u \neq v} \sum_k p_{t,k}^{(u)} \log p_{s,k}^{(v)}. \quad (2.42)$$

The teacher is not optimized by backpropagation. Its parameters are an exponential moving average of the student parameters,

$$\xi \leftarrow \mu \xi + (1 - \mu) \theta, \quad 0 < \mu < 1. \quad (2.43)$$

Centering, temperature sharpening, and the slowly changing teacher prevent the trivial solution in which every image receives the same representation. DINO features transfer well, and attention maps can align with object regions even without segmentation labels.

Masked autoencoders (MAE) randomly divide patch indices into visible and masked sets $\mathcal{V}$ and $\mathcal{M}$ [52]. Only the visible patches are processed by the expensive encoder,

$$H_{\mathcal{V}} = f_{\text{enc}}(I_{\mathcal{V}}).$$

A lightweight decoder receives H_V , position information, and learned mask tokens, and predicts a vector $\hat{x}_j$ for each missing patch. With $x_j = \text{vec}(I_j)$ and patchwise normalization $\text{norm}(\cdot)$, the reconstruction objective is

$$\mathcal{L}_{\text{MAE}} = \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{M}} \|\hat{x}_j - \text{norm}(x_j)\|_2^2. \quad (2.44)$$

Removing masked patches before the encoder reduces pretraining computation, while the decoder learns to combine visible context with spatial position to recover missing content.

After any of these pretraining procedures, the prediction head or lightweight decoder can be discarded and the encoder retained as a reusable backbone. Classification attaches a global readout. Detection and segmentation instead use spatial patch features, often from several resolutions, and attach task-specific decoders. The choice of pretraining objective therefore affects which information is most readily available during downstream adaptation.

The same pattern appears in multimodal models. A broad paired dataset is used to learn representations, and the model is later transferred to classification, retrieval, question answering, or generation. CLIP [127] learns image and text encoders with a contrastive objective and can then classify images by comparing image embeddings with text embeddings of candidate labels. Fusion and generative multimodal models use related pretrain and transfer patterns, but they allow token level interaction between modalities. Section 2.1.6 discusses these architectures in more detail.

2.1.4 One Dimensional Sequence Applications

One dimensional Transformer applications differ in both the input representation and the required output. Sequence classification maps a full input to one label. Token classification predicts a label for every position. Sequence transduction maps an input sequence to a different output sequence. Autoregressive modeling predicts a continuation. These tasks use similar Transformer layers but require different masks, pooling operations, and output heads.

For sequence classification, a model can prepend a learned summary token and use its final representation, as in BERT [35]. Mean pooling is another option when no position should have a privileged role. Token classification instead applies the same prediction head to every final token representation. Named entity recognition, residue labeling, and frame level classification have this form. Sequence transduction uses an encoder-decoder or a decoder conditioned on a serialized input.

Language. Language is the original application domain of the Transformer. In autoregressive language modeling, the input is a prefix and the target is the next token. During training, predictions for all positions can be evaluated in parallel under a triangular causal mask. During generation, tokens are produced one at a time. GPT-2 showed that a single decoder-only model trained on a large and diverse text corpus can perform many language tasks with no task specific training [124]. GPT-3 extended this to in-context learning. The model performs a new task from an instruction and a few examples placed in its context window, without any parameter update [15]. The quality of these models improves in a predictable way as model size, data, and compute increase, a relationship captured by empirical scaling laws [65]. Language modeling benchmarks include character level corpora such as enwik8 [101] and word level corpora such as WikiText-103 [106]. Autoregressive modeling is one of the settings used to evaluate the one dimensional construction in Chapter 3.

This autoregressive execution pattern makes the context-window length a central resource. The context window defines the evidence available for each prediction. A longer context can contain more demonstrations, earlier conversation turns, or distant document sections. It also increases the key-value cache during generation. The model must decide which earlier tokens remain relevant as the context grows. Local syntax often depends on nearby tokens, while document structure and reference resolution can require long range interactions. An efficient method should retain both scales.

Bidirectional encoders address a different problem. Their purpose is to build contextual token representations rather than to generate text from left to right. Masked language modeling hides a fraction of the tokens and trains the model to predict them from context on both sides. BERT showed that such a pretrained encoder transfers to classification, named entity recognition, retrieval, and question answering through light finetuning [35]. RoBERTa later showed that the training recipe matters as much as the architecture. With longer training, more data, and larger batches, the same model reaches substantially better results [97]. Bidirectional masked language modeling is the second one dimensional setting used in Chapter 3.

Bidirectional attention is useful when the complete input is available before prediction. A document classifier can use evidence anywhere in the document. A token representation can combine left and right context. Retrieval systems can encode a query and a candidate into vectors for similarity search, or jointly encode the pair for more detailed interaction. These settings do not require causal masking, but padding masks are still needed when variable length sequences are batched.

Encoder-decoder Transformers address sequence transduction. Machine translation is

the original example [159]. Cross attention gives decoder tokens access to encoder representations of the input, so the output can depend on the full source sequence. T5 casts many language tasks in a single text-to-text form, where both the input and the output are text strings [128]. This lets one model handle translation, summarization, classification, and question answering with the same objective. BART pretrains an encoder-decoder by corrupting text and learning to reconstruct it, which suits generation tasks such as summarization [87]. Long inputs are useful in these tasks because relevant evidence can occur far from the output position.

Let n_s and n_t be the source and target lengths. Encoder self attention costs $\mathcal{O}(n_s^2 d)$, decoder self attention during training costs $\mathcal{O}(n_t^2 d)$, and cross attention costs $\mathcal{O}(n_s n_t d)$. Their attention-score memories scale respectively as $\Theta(n_s^2)$, $\Theta(n_t^2)$, and $\Theta(n_s n_t)$ per head. Summarization of a long document can therefore be expensive even when the output is short. Translation usually has shorter inputs, but it requires accurate alignment between source and target positions.

Biology and Scientific Sequences. A protein is a chain of amino acids, and many of its properties depend on interactions between residues that are distant along the chain. AlphaFold predicts protein structure with attention based components. Its Evoformer module applies attention over a multiple sequence alignment and over a pairwise residue representation, so that evolutionary and geometric information are exchanged before structure is predicted [61]. Protein language models take a different route. ESM trains a masked language model on hundreds of millions of protein sequences and learns representations that reflect secondary structure, residue contacts, and function, from sequence data alone [130]. The analogy with language is direct. Residues play the role of tokens, and evolutionary variation plays the role of context.

Across both structure-prediction systems such as AlphaFold and protein language models such as ESM, biological sequences differ from natural language in several ways. Their alphabets are small, sequences can be very long, and labels are expensive because they depend on experiments. Self-supervised learning is therefore especially useful. Local motifs can determine binding or secondary structure, while distant positions can interact after a protein folds. This combination of local motifs and distant effects resembles the near field and far field structure used in Chapter 3.

Genomic sequences are a related domain. DNA and RNA are long strings over a small alphabet, and regulatory elements can act over large distances. Enformer uses attention to integrate long range context and improves prediction of gene expression from sequence [6]. DNABERT adapts masked language modeling to nucleotide sequences and transfers to tasks such as promoter and splice site prediction [59]. Long range interaction is the main

reason attention is attractive here, because an important signal can lie far from the position being predicted.

Audio and Music. Audio is also sequential. wav2vec 2.0 learns speech representations from unlabeled audio. It masks spans of a latent feature sequence and solves a contrastive task that identifies the correct quantized latent among distractors. The pretrained model is then finetuned for recognition with limited labeled data [7]. Whisper takes a supervised route. It trains an encoder-decoder on a large and weakly labeled multilingual corpus, which gives robust speech recognition and translation across many conditions [126]. Conformer combines attention with convolution inside each block, so the model captures global context and local acoustic detail at the same time [45].

Raw audio has a much higher sampling rate than text token sequences. Speech systems therefore use convolutional subsampling, spectrogram frames, or learned latent units before attention. This reduces sequence length and adds a local acoustic bias. Recognition then maps many input frames to a shorter output token sequence. The mismatch in lengths makes encoder-decoder and alignment based objectives natural choices.

Music is a further example of long range symbolic sequence structure. Music Transformer generates music as a sequence of note events. It uses relative position information so that repeated motifs and long range musical structure can be represented [58]. The same idea applies to other event sequences where structure repeats at several time scales.

Decision Making and Control. Transformers also model sequences of decisions. Decision Transformer treats reinforcement learning as sequence modeling. It forms a single sequence from past returns, states, and actions, and predicts the next action with a causal decoder. Conditioning on a target return lets the model produce actions that aim for a desired outcome [22]. This shows that the sequence abstraction extends beyond perception and language to control.

Time-Series Forecasting. Time series forecasting operates on real valued observations rather than discrete tokens [190]. Each time step is projected to an embedding, often together with calendar variables, an observation mask, and a time interval. Regular sampling gives a fixed grid. Irregular sampling requires the model to represent elapsed time explicitly. Forecasting can be autoregressive, where predictions are fed back one step at a time, or direct, where a block of future values is predicted together. Long histories help represent trends and seasonality but also increase cost.

Source Code. Source code has both sequential order and a latent tree structure. A program contains identifiers, scopes, repeated definitions, and dependencies that may span many tokens. CodeBERT pretrains on paired natural language and code and supports

search, summarization, and completion [39]. Models can also include syntax or data flow information, but the token sequence remains a common interface.

These applications place different demands on attention. Language and code require order and causal masking for generation. Biological sequences require very long receptive fields. Audio requires strong local processing before global interaction. Time series may include continuous values and irregular time intervals. Efficient attention must preserve the structure needed by the application rather than only reduce an abstract matrix cost. Symbolic regression inputs are unordered sets rather than one dimensional sequences and are therefore reviewed separately in Section 2.4.3.

2.1.5 Vision and Imaging Applications

Vision Transformers support image classification, object detection, segmentation, image generation, and video understanding. These tasks can share visual backbones, but they require different output representations and different amounts of spatial detail.

Image Classification. Image classification summarizes the patch-token sequence with a class token or mean pooling and maps the resulting representation to one label. The ViT formulation and its visual tokenization are described in Section 2.1.2.

Object Detection. These multiscale backbones provide the spatial features needed by task specific prediction heads. Object detection uses them to predict the locations and classes of several objects in an image. Detection Transformer reformulates this as direct set prediction. A fixed set of learned object queries attends to image features through an encoder-decoder Transformer, and each query predicts one object or a no-object label. A bipartite matching between predictions and ground truth objects defines the training loss, which removes hand designed steps such as anchor boxes and non-maximum suppression [19]. The original model converges slowly and handles small objects poorly. Deformable DETR addresses this by letting each query attend to a small set of sampled locations across several feature scales, which speeds up training and improves detection of small objects [191].

The object queries in DETR are output slots rather than image locations. Their predictions are permutation invariant as a set, so training must decide which prediction corresponds to which ground truth object. Bipartite matching supplies this assignment. Multiscale features remain important because a small object may occupy only a few fine resolution tokens, while a large object is easier to identify from coarse context. Deformable attention combines these scales without asking every query to inspect every spatial loca-

tion. Segmentation uses the same spatial features but asks for a finer output at every pixel.

Segmentation. Segmentation assigns a label to each pixel. Semantic segmentation labels each pixel with a class. Instance and panoptic segmentation additionally separate individual objects. SegFormer pairs a hierarchical Transformer encoder with a lightweight decoder and reports strong results on standard benchmarks [175, 189]. Mask2Former treats segmentation as mask classification. It predicts a set of binary masks together with a label for each, and a single architecture then handles semantic, instance, and panoptic segmentation [25].

Dense prediction requires a different readout from classification. A class token can summarize whether an object is present, but it cannot by itself preserve the location of every boundary. Segmentation systems therefore retain a grid of patch features, combine several resolutions, and decode them to a fine output map. Skip connections or feature pyramids restore local detail that would otherwise be lost during patch merging. Segmentation has also moved toward large promptable models. Segment Anything trains on a very large dataset of masks and produces a segmentation for a user supplied prompt such as a point or a box. An image encoder computes reusable visual features. A prompt encoder represents points, boxes, or masks, and a lightweight mask decoder combines the two representations. The model can segment objects it was not explicitly trained to recognize, which makes it a general purpose component in image pipelines [68].

Image Generation. The applications above use image features to predict classes, boxes, or masks. A Transformer can also model an image distribution and generate new samples. In a diffusion model, a network repeatedly predicts how to remove noise from a corrupted sample. Diffusion Transformer replaces the convolutional denoising backbone with a Transformer that operates on latent image patches. The diffusion time step and optional class label condition the Transformer blocks. Computation takes place in a compressed latent space, so the token count is lower than it would be at full pixel resolution. Image quality improves as the Transformer is scaled [115].

Video. Video extends the patch-based image formulation along a temporal axis. A clip is divided into space-time tubes or into patches from individual frames. For T frames and patch dimensions p_t , p_h , and p_w , the token count is

$$n_{\text{video}} = \frac{THW}{p_t p_h p_w}. \quad (2.45)$$

Motion requires the model to connect visual evidence across time, while appearance recognition still requires spatial interaction within each frame. Full space-time attention is

expensive because its matrix is quadratic in the product of temporal and spatial token counts. TimeSformer separates attention over the spatial and temporal axes, which keeps the cost manageable while modeling motion [12]. ViViT studies several ways to factorize space-time attention for video classification [5]. Longer clips improve temporal coverage but increase both training memory and inference latency.

Two dimensional applications also expose a distinction between local detail and global context. Texture, edges, and boundaries are local. Object identity, scene layout, and interactions between distant regions require a larger receptive field. Convolution handles local structure efficiently but expands its receptive field gradually. Dense attention provides immediate global interaction but pays a quadratic cost. Hierarchical vision Transformers combine local processing at fine resolution with broader interaction after spatial reduction.

The two dimensional construction in Chapter 3 addresses the same problem from a multilevel viewpoint. Nearby patches interact at full resolution. Distant regions interact through learned summaries on a quadtree. The goal is to preserve a global receptive field while avoiding dense all-pairs attention on the full image grid.

2.1.6 Multimodal Transformers and Contrastive Learning

A multimodal model processes more than one data modality. Common examples pair text with images, video, audio, or structured scientific observations [2, 100, 127]. Each modality is converted into embedded tokens or latent vectors before attention is applied.

There are three common architectural patterns. A dual encoder processes each modality separately and compares global representations, as in CLIP [127]. A fusion encoder permits token-level self attention or cross attention between modalities, as in ViLBERT [100]. A generative model conditions a decoder on one or more input modalities, as in Flamingo [2]. The patterns can be combined.

Contrastive learning is a standard objective for dual encoders. Suppose a minibatch of size B contains matched pairs (a_i, b_i) from two modalities. Let z_i^a and z_i^b be normalized encoder outputs and let $\tau > 0$ be a temperature. One direction of the contrastive loss is

$$\mathcal{L}_{a \rightarrow b} = -\frac{1}{B} \sum_{i=1}^B \log \frac{\exp((z_i^a)^\top z_i^b / \tau)}{\sum_{j=1}^B \exp((z_i^a)^\top z_j^b / \tau)}. \quad (2.46)$$

A symmetric objective averages $\mathcal{L}_{a \rightarrow b}$ and $\mathcal{L}_{b \rightarrow a}$. Matching pairs are pulled together in representation space. Nonmatching pairs in the batch act as negatives. SimCLR studies

this objective for different augmented views of the same image [24]. CLIP applies a symmetric contrastive objective to image-text pairs collected from the web [127]. The learned representation supports zero-shot image classification and transfer to many visual tasks.

Fusion models provide more detailed interaction between modalities. ViLBERT extends BERT to a two-stream image-text model with co-attentional Transformer layers [100]. Separate visual and textual streams exchange information during pretraining and can be transferred to visual question answering, referring expressions, and image retrieval. BLIP combines understanding and generation objectives for image-text pretraining [88]. Flamingo bridges pretrained vision and language models and processes interleaved image and text inputs for few-shot adaptation [2].

Dual encoders and fusion models make different computational tradeoffs. A dual encoder can compute and store one embedding per item, which supports efficient retrieval over a large collection. It does not model fine token level interaction between a query and each candidate. A fusion model can represent those interactions but must process the paired inputs jointly. Generative multimodal models add a further requirement. They must preserve causal masking in the decoder while exposing visual or other conditioning information through cross attention or inserted latent tokens.

The symbolic regression problem of Chapter 4 has a related multimodal structure. One view of a function is symbolic. It is represented by an expression tree or a token sequence. Another view is numerical. It is represented by sampled input-output pairs. SNIP uses contrastive learning to align symbolic and numerical views [141]. The project in Chapter 4 also uses a Transformer to extract useful structure from numerical observations. Image-text models and symbolic-numeric models do not have identical architectures, but they use the same broad pretraining idea. Paired views provide supervision for a representation that can later be adapted to a downstream task.

2.2 Efficient Transformers and Attention

This section reviews dense-attention cost and six alternatives: sparse attention [11, 26], routing [69, 133], linearized attention [27, 66], pooling and downsampling [96, 170], hierarchical attention [111, 192], and IO-aware exact attention [30, 32]. These methods provide background for Chapter 3.

2.2.1 The Cost of Dense Global Attention

The product $QK^\top$ in Equation (2.5) contains n^2 entries. The subsequent softmax and multiplication with V have the same asymptotic cost. Self attention therefore requires $\mathcal{O}(n^2\hat{d})$ time and $\mathcal{O}(n^2)$ memory per layer for an input of length n . This quadratic dependence is the central computational bottleneck of the architecture. It limits the maximum sequence length feasible in practice [149, 150].

For a one dimensional sequence with h attention heads, explicitly storing the attention scores requires an $h \times n \times n$ tensor. Doubling the context length therefore multiplies score computation and storage by four. A causal mask removes future-token interactions conceptually, but a conventional dense training implementation still evaluates a triangular portion of an $n \times n$ score matrix and retains quadratic scaling. This is especially costly during training because intermediate activations are needed for backpropagation. During autoregressive inference, the key-value cache grows linearly with context length, but each newly generated query still compares with all cached keys.

The same issue becomes more severe when a two dimensional image is represented by a fine patch grid. For an image of height H_0 , width W_0 , and patch dimensions $p_h \times p_w$, the number of visual tokens and the dense-attention interaction cost are

$$n_{\text{img}} = \frac{H_0 W_0}{p_h p_w}, \quad \mathcal{C}_{\text{img}} = \mathcal{O}\left(\frac{H_0^2 W_0^2}{p_h^2 p_w^2} \hat{d}\right) = \mathcal{O}(n_{\text{img}}^2 \hat{d}). \quad (2.47)$$

A 224×224 image with 16×16 patches has 196 patch tokens and $196^2 = 38,416$ query-key scores per head. At a resolution of 1024×1024 with the same patch size, the sequence contains 4096 tokens and one head contains more than 16 million scores. Dense prediction often uses still finer intermediate feature grids to preserve small objects and boundaries. If the feature map itself has spatial size $H \times W$, then $n = HW$ and dense attention costs $\mathcal{O}(H^2 W^2 \hat{d})$ [171, 175].

Video adds a temporal factor to this spatial grid. Using the token count n_{video} in Equation (2.45), full space-time attention costs $\mathcal{O}(n_{\text{video}}^2 \hat{d})$. Increasing the number of frames, reducing the patch size, or increasing the image resolution therefore has a multiplicative effect on cost. These scaling properties motivate local windows, spatial downsampling, factorized space-time attention, and multilevel representations.

In a fusion model with n_t text tokens and n_v visual tokens, joint self attention costs $\mathcal{O}((n_t + n_v)^2 \hat{d})$; cross attention alone costs $\mathcal{O}(n_t n_v \hat{d})$ [100].

Sparse-attention methods assume that only a subset of token pairs needs direct interaction [11, 26, 186]. Other methods approximate the attention matrix or compute exact attention with less data movement. The following subsections separate these strategies.

The relevant cost depends on the stage of the model. Pretraining is sensitive to activation memory because gradients must be retained or recomputed. Finetuning adds optimizer state and may update only a small adapter. Autoregressive inference is often limited by memory bandwidth and by the key-value cache. An efficient attention method should state which cost it reduces. Lower asymptotic arithmetic cost, lower memory use, and lower wall clock time are related but not interchangeable claims.

2.2.2 Sparse and Local Attention

The simplest efficient attention strategy restricts each query to attend to a fixed subset of keys. This yields a sparse attention matrix with $\mathcal{O}(n)$ or $\mathcal{O}(n\sqrt{n})$ nonzero entries. Concretely, a mask $M \in \{0, -\infty\}^{n \times n}$ with $M_{ij} = 0$ on an allowed edge set E and $M_{ij} = -\infty$ otherwise gives masked attention

$$\text{Attn}_M(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{\hat{d}}} + M\right) V,$$

with time and memory cost $\mathcal{O}(|E|\hat{d})$. The design choice for sparse attention is the structure of E .

Local or sliding window attention keeps only a band around the diagonal of the attention matrix. Each token attends to a fixed window of neighbours. This pattern is simple to implement, has linear complexity, and matches the locality bias of many sequence tasks. Sparse Transformer [26] combines local windows with strided patterns that allow longer range interactions. Longformer [11] adds global tokens that can attend everywhere. BigBird [186] combines local, global, and random attention. The resulting pattern is a universal approximator of full attention. ETC [1] adopts a similar global plus local structure tailored to structured inputs. The earlier work of Liu et al. [96] applies memory compressed attention to generate Wikipedia articles from long sequences.

In vision, local window attention takes the form of attending within fixed spatial patches. Axial attention [54] restricts attention to rows and columns of the spatial grid separately. This reduces the cost from $\mathcal{O}((HW)^2)$ to $\mathcal{O}(HW(H + W))$. HaloNet [161] attends within blocks plus a small halo region. Swin Transformer [99] attends within nonoverlapping windows and shifts the window grid between consecutive layers to enable cross window communication. The Neighborhood Attention Transformer [49] replaces the shifted window with a local neighborhood around each query.

More recent work explores window shapes that span larger portions of the image. CSWin Transformer [37] uses cross shaped stripes that span the full image width or height.

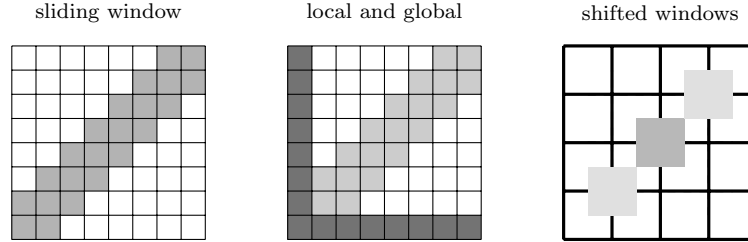


Figure 2.3: Representative fixed sparse patterns. A sliding window produces a diagonal band; global tokens add full rows and columns; shifted spatial windows connect neighboring regions across consecutive layers.

DiagSWin [90] extends this idea to diagonal shaped stripes for object detection and segmentation. These methods enlarge the receptive field per layer through geometric window design while retaining linear complexity in the number of patches.

Sparse patterns are simple when their geometry is fixed. For example, a width- w sliding window has $\mathcal{O}(nw)$ edges and can be implemented as a banded operation. Interactions outside the pattern require multiple layers, and the pattern may be task specific. Global tokens shorten long paths but retain a fixed sparsity design.

The connectivity graph gives a useful way to compare sparse patterns. Tokens are vertices and allowed attention edges are directed links. A local window has bounded degree but a graph diameter that grows with sequence length. Adding strided edges, shifted windows, or global tokens shortens paths between distant positions. This improves communication without returning to all pairs attention. The graph view also explains why a sparsity pattern that is effective for a document may not transfer directly to an image grid.

2.2.3 Clustering and Routing Attention

Routing methods use data-dependent sparse patterns. Reformer [69] hashes similar query and key vectors into common buckets. Routing Transformer [133] uses online k -means. Clustered Attention [167] attends to cluster centroids. ClusterFormer [168] and the method of Wang et al. [169] use related assignments. These methods adapt the connectivity to the input, but clustering and load balancing add overhead.

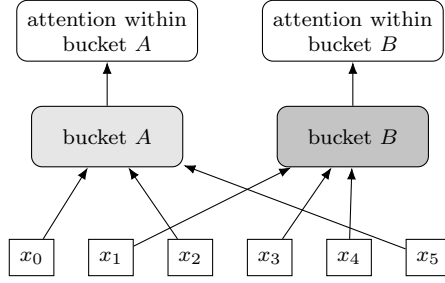


Figure 2.4: Data-dependent routing attention. Similar queries and keys are assigned to common buckets, and dense attention is evaluated only within each bucket.

2.2.4 Linearized Attention

Linearized attention avoids the $n \times n$ matrix by factorizing the attention kernel. Suppose a nonnegative map ϕ approximates the exponential kernel. Define

$$S = \phi(K)^\top V, \quad z = \phi(K)^\top \mathbf{1}_n. \quad (2.48)$$

The normalized output for query q_i is then

$$y_i = \frac{\phi(q_i)^\top S}{\phi(q_i)^\top z}. \quad (2.49)$$

The summaries S and z are reused by all queries. Linear Transformer [66] uses $\phi(x) = \text{ELU}(x) + 1$. Performer [27] and Random Feature Attention [116] use random features with approximation bounds. cosFormer [122] adds a cosine reweighting. The cost is $\mathcal{O}(n\hat{d}^2)$ when the feature dimension is comparable to $\hat{d}$.

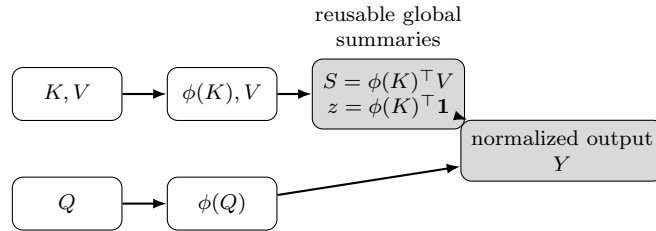


Figure 2.5: Linearized attention changes the multiplication order. Keys and values are first compressed into summaries S and z , which are then reused for every transformed query.

Linearization is linear in n , but causal cumulative sums are sequential and the kernel approximation may be poor for sharply concentrated attention [149, 150].

2.2.5 Pooling and Downsampling

Downsampling reduces the number of keys and values. Let $S_K, S_V \in \mathbb{R}^{k \times n}$ with $k \ll n$. Then

$$\tilde{K} = S_K K, \quad \tilde{V} = S_V V, \quad Y = \text{softmax}\left(\frac{Q\tilde{K}^\top}{\sqrt{\hat{d}}}\right) \tilde{V}. \quad (2.50)$$

The attention matrix is $n \times k$, with cost $\mathcal{O}(nk\hat{d})$. Average pooling over groups $G_1, \dots, G_k$ is the special case

$$\tilde{K}_a = \frac{1}{|G_a|} \sum_{j \in G_a} K_j, \quad \tilde{V}_a = \frac{1}{|G_a|} \sum_{j \in G_a} V_j. \quad (2.51)$$

Memory-compressed attention uses strided convolutions [96]; Linformer learns downsampling maps [170].

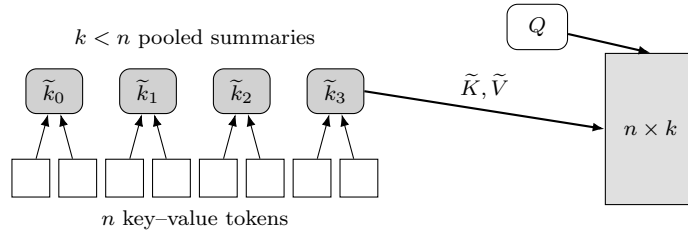


Figure 2.6: Pooling and downsampling reduce the key-value sequence before attention. The resulting interaction matrix is rectangular, with n queries and only k pooled keys.

Vision models use this approach because patch grids admit spatial pooling. PVT merges patches across stages [171]; P2T pools keys and values at several scales [174]; LRFormer uses a fixed low-resolution grid [188]. HAT-Net [98] and Focal Self Attention [178] retain fine local features and coarser distant summaries.

Downsampling is simple and changes the matrix from $n \times n$ to $n \times k$. Shared summaries may discard location or small-scale detail. Region-specific and multiscale summaries preserve more geometry but use more memory. The taxonomy follows the dominant operation; hybrid methods can belong to more than one family [150].

FMA differs from these methods in two ways. It keeps direct near-field interactions and uses several levels of learned far-field summaries. It also organizes summaries by a fixed spatial hierarchy rather than one global downsampling map.

2.2.6 Hierarchical Attention

Hierarchical attention represents nearby and distant interactions at different resolutions. Nearby tokens are kept separate because their interactions may depend on fine detail. Distant groups are represented by summaries because the effect of every individual token is often less important there. A hierarchy repeats this compression at several scales, so information can cross the full sequence without materializing every entry of the $n \times n$ attention matrix. The main design choices are how to construct the summaries, which blocks are evaluated at each level, and whether the hierarchy is fixed or chosen from the input.

H-Transformer-1D [192] uses a fixed binary tree over the sequence. Let $\tilde{Q}^{(0)} = Q$, $\tilde{K}^{(0)} = K$, and $\tilde{V}^{(0)} = V$. Adjacent representations are coarsened recursively as

$$\begin{aligned}\tilde{Q}_j^{(\ell+1)} &= \frac{1}{2} \left(\tilde{Q}_{2j}^{(\ell)} + \tilde{Q}_{2j+1}^{(\ell)} \right), \\ \tilde{K}_j^{(\ell+1)} &= \frac{1}{2} \left(\tilde{K}_{2j}^{(\ell)} + \tilde{K}_{2j+1}^{(\ell)} \right), \\ \tilde{V}_j^{(\ell+1)} &= \tilde{V}_{2j}^{(\ell)} + \tilde{V}_{2j+1}^{(\ell)}.\end{aligned}\tag{2.52}$$

Averaging the queries and keys gives a representative location in feature space, while summing the values preserves the aggregate contribution of a group. At the finest level, the method evaluates diagonal blocks that contain interactions within nearby token groups. At each coarser level it evaluates only neighboring off-diagonal blocks that were not already represented at a finer level. Thus a pair of nearby tokens interacts at high resolution, whereas a pair of well-separated groups interacts through coarsened queries, keys, and values. Figure 2.7 illustrates this construction.

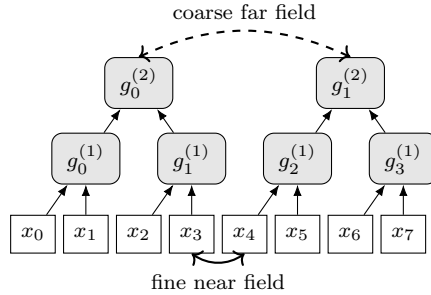


Figure 2.7: Binary hierarchy in H-Transformer-1D. Nearby interactions are represented by fine tokens, while distant groups interact through coarser summaries.

If the leaf block size is fixed, every level contains a constant amount of work per active block and the number of blocks decreases geometrically. A schematic cost sum is

$$\sum_{\ell=0}^{L_h-1} \mathcal{O}\left(\frac{n}{2^\ell} b \hat{d}\right) = \mathcal{O}(n b \hat{d}), \quad (2.53)$$

where b is the fixed block width. This gives linear scaling in n . The fixed hierarchy is regular and easy to batch, but it assumes that sequence distance is a useful guide to interaction resolution. A strong long-range interaction is still compressed if its tokens are far apart in the binary partition.

Multi-resolution attention [187] instead constructs an adaptive partition of the attention matrix. Let A denote the nonnegative attention kernel before row normalization, and let $B_{x,y}^{(s)} = I_x^{(s)} \times I_y^{(s)}$ be a square block whose query and key index sets have width s . Its block average is

$$\mu_{s,x,y} = \frac{1}{|B_{x,y}^{(s)}|} \sum_{(i,j) \in B_{x,y}^{(s)}} A_{ij}. \quad (2.54)$$

The approximation initially assigns one value $\mu_{s,x,y}$ to every entry of a coarse block. It then scores the current blocks, selects the most important ones, and replaces each selected block by its children at the next finer resolution. Repeating this operation produces a nonuniform collection of blocks: smooth or weak regions remain coarse, while regions containing strong or rapidly changing interactions receive more computation. After the selected blocks have been evaluated, the approximate kernel is normalized row by row.

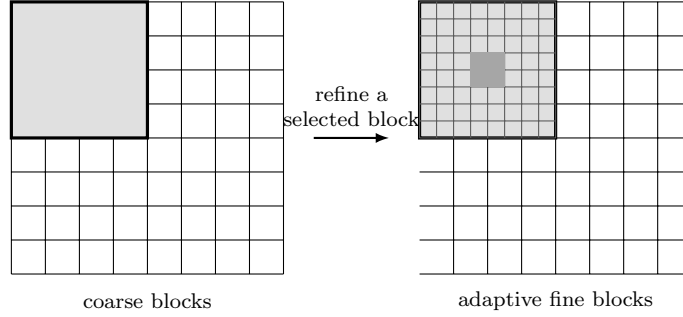


Figure 2.8: Adaptive block refinement in multi-resolution attention. Most of the attention matrix remains coarse, while selected regions are subdivided and evaluated at finer resolution.

Suppose the initial block width is s_0 , the subsequent widths are $s_1, \dots, s_L$, and m_i blocks are refined at level i . The block construction has the form

$$\mathcal{O}\left(n + \left(\frac{n}{s_0}\right)^2 + \sum_{i=1}^L m_i \left(\frac{s_{i-1}}{s_i}\right)^2\right). \quad (2.55)$$

The refinement budgets m_i expose a direct accuracy–cost tradeoff. This adaptivity can preserve isolated long-range interactions that a fixed spatial hierarchy would compress. Its cost pattern is less regular, however, because the selected blocks and their resolutions vary across inputs.

FMMformer [111] applies the near-field/far-field principle more directly. It approximates the attention operator by a banded local term and several low-rank global terms. If $\text{band}_k(\cdot)$ removes entries outside a bandwidth k , the near-field output is computed as

$$Y_N = \text{softmax}\left(\text{band}_k\left(\frac{QK^\top}{\sqrt{\hat{d}}}\right)\right)V.$$

This is ordinary attention restricted to nearby keys. For far-field component r , let ϕ_r be a feature map and form the reusable summaries $S_r = \phi_r(K)^\top V$ and $z_r = \phi_r(K)^\top \mathbf{1}_n$, analogous to Equations (2.48)–(2.49). The far-field output and the complete approximation are

$$Y_F = \sum_{r=1}^R \frac{\phi_r(Q)S_r}{\phi_r(Q)z_r}, \quad \hat{Y} = Y_N + Y_F,$$

where each denominator is applied rowwise. The bandwidth k controls how much local structure is represented exactly, while the number and dimension of the feature maps control the capacity of the far field. For feature width p , the cost is $\mathcal{O}(nkd + nRp\hat{d})$, which is linear in n when k , R , and p are fixed.

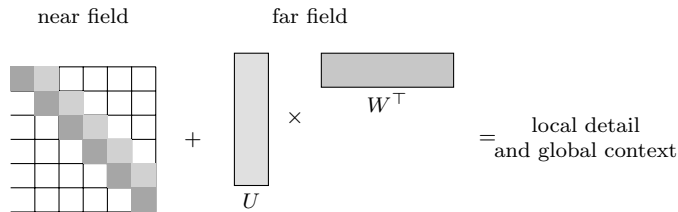


Figure 2.9: Near-field/far-field decomposition in FMMformer. A banded matrix retains local interactions, and low-rank factors communicate information over the full sequence.

FMMformer is grouped with hierarchical methods because its decomposition is motivated by fast multipole and hierarchical-matrix ideas, although its practical far field is a global low-rank representation rather than a full tree of scale-specific bases. H-Transformer-1D provides a regular multilevel partition, multi-resolution attention provides data-dependent refinement, and FMMformer provides an explicit direct-plus-low-rank split. These distinctions matter when the data contain both sharp local interactions and structured long-range interactions at several spatial scales.

Chapter 3 combines these principles in a learned multilevel operator. It keeps direct near-field interactions, attaches separate learned far-field bases to multiple tree levels, supports autoregressive and bidirectional masks, and extends the one dimensional hierarchy to a two dimensional quadtree. The construction therefore retains the regularity of a fixed hierarchy while giving the far field more scale-dependent capacity than a single global low-rank term.

2.2.7 IO-Aware Exact Attention

IO-aware methods reduce data movement without approximating attention. FlashAttention and FlashAttention-2 tile the softmax computation and avoid storing the full matrix [30, 32]. Their arithmetic cost remains $\mathcal{O}(n^2)$, but memory traffic and wall-clock time decrease. Ring Attention distributes exact blockwise attention across devices and supports million-token contexts [95].

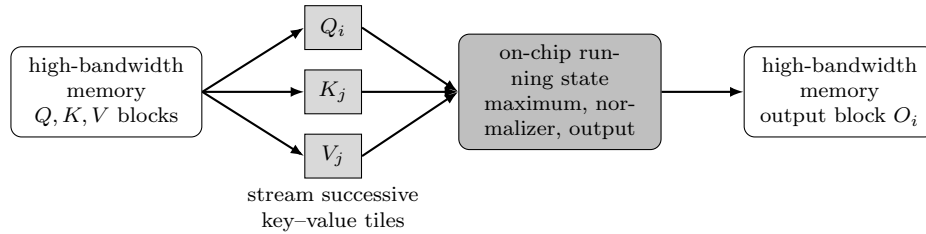


Figure 2.10: IO-aware exact attention processes the score matrix tile by tile. Only small query, key, and value blocks and a running softmax state reside in fast on-chip memory, so the full $n \times n$ matrix is never written to external memory.

Recent methods reach longer contexts with different compromises. LongNet uses dilated sparse attention and is designed to scale beyond one billion tokens [36]. Infini-attention combines local attention with compressive memory and evaluates one-million-token retrieval and 500,000-token summarization [110]. InfiniteHiP uses hierarchical token prun-

ing and reports three-million-token inference on one GPU [84]. These limits depend on hardware, training, and task, so they are not directly comparable.

No method dominates all settings. Exact IO-aware kernels preserve dense attention; sparse and pooled methods exploit structure; hierarchical methods retain local detail and coarse global communication.

2.3 Hierarchical and Multilevel Numerical Methods

This section reviews the numerical ideas that directly motivate Fast Multipole Attention. The focus is not a broad survey of multilevel solvers. It is the specific pattern that Chapter 3 turns into a neural attention operator. An all pairs interaction is split into a near field and a far field. The near field is kept at full resolution. The far field is compressed through summaries arranged on a tree. This is the central idea behind the Fast Multipole Method and hierarchical matrices, and it is the design principle used by FMA.

2.3.1 All Pairs Sums and the Fast Multipole Method

The n body problem is the classical setting for fast hierarchical summation. Given particles at positions $x_1, \dots, x_n$ with charges or masses $q_1, \dots, q_n$, one wants to evaluate

$$\phi(x_i) = \sum_{j \neq i} \frac{q_j}{\|x_i - x_j\|} \quad (2.56)$$

for every i . A direct evaluation forms every pair interaction and costs $\mathcal{O}(n^2)$. In the Transformer, dense self attention has the same all pairs shape. It forms a score between every query and every key and then applies the resulting weights to values.

It is useful to write a more general kernel sum,

$$u_i = \sum_{j=1}^n K(x_i, y_j) w_j, \quad i = 1, \dots, n. \quad (2.57)$$

Here u_i is the accumulated kernel response at target x_i . It has the same dimension as the source weights w_j . The targets are x_i , the sources are y_j , and the source weights are w_j . In the Transformer, dense attention is not a fixed physical kernel, but it has the same

computational form. For one attention head, let $\mathbf{q}_i, \mathbf{k}_j \in \mathbb{R}^{\hat{d}}$ be the query and key vectors and let $\mathbf{v}_j \in \mathbb{R}^{d_v}$ be the value vector. The output at query i is

$$\mathbf{o}_i = \sum_{j=1}^n \alpha_{ij} \mathbf{v}_j, \quad \alpha_{ij} = \frac{\exp(\mathbf{q}_i^\top \mathbf{k}_j / \sqrt{\hat{d}})}{\sum_{\ell=1}^n \exp(\mathbf{q}_i^\top \mathbf{k}_\ell / \sqrt{\hat{d}})}. \quad (2.58)$$

Thus $\mathbf{o}_i$ is a vector valued kernel sum in which the normalized data dependent attention weight α_{ij} plays the role of $K(x_i, y_j)$ and the value vector $\mathbf{v}_j$ plays the role of w_j .

The fast multipole method of Greengard and Rokhlin [42] reduces the cost of kernel sums by using geometry. Nearby sources are evaluated directly. Distant sources are grouped into clusters. For many smooth kernels, if a target region and a source region are well separated, the kernel can be approximated by a separated expansion

$$K(x, y) \approx \sum_{m=1}^p a_m(x) b_m(y), \quad (2.59)$$

where the functions a_m depend on the target variable and the functions b_m depend on the source variable. If C is a source cluster, its source moments are

$$M_m(C) = \sum_{j \in C} b_m(y_j) w_j, \quad m = 1, \dots, p. \quad (2.60)$$

Then a far field contribution from all sources in C to a target x_i can be approximated by

$$u_i^{(C)} := \sum_{j \in C} K(x_i, y_j) w_j \approx \sum_{m=1}^p a_m(x_i) M_m(C). \quad (2.61)$$

Here $u_i^{(C)}$ denotes the part of the response at x_i generated by the sources in C . If $\mathcal{N}_i$ is the set of nearby sources evaluated directly and $\mathcal{F}_i$ is the collection of well separated source clusters, the complete response is approximated by

$$u_i \approx \sum_{j \in \mathcal{N}_i} K(x_i, y_j) w_j + \sum_{C \in \mathcal{F}_i} \sum_{m=1}^p a_m(x_i) M_m(C). \quad (2.62)$$

The important point is that the inner sum over sources in Equation (2.60) is computed once per cluster and reused for many targets. The direct pair count between a target region and a source cluster is replaced by p summary coefficients. The expansion order p controls the accuracy and the cost.

An FMM computation is organized around a tree [102]. The upward pass builds summaries from leaves to the root. The interaction pass uses summaries for well separated source and target clusters. The downward pass transfers accumulated far field information back to fine target regions. Direct interactions between nearby leaves are added separately. This separation is what FMA borrows. Local information should remain exact. Distant information should be summarized and reused. Figure 2.11 illustrates these stages.

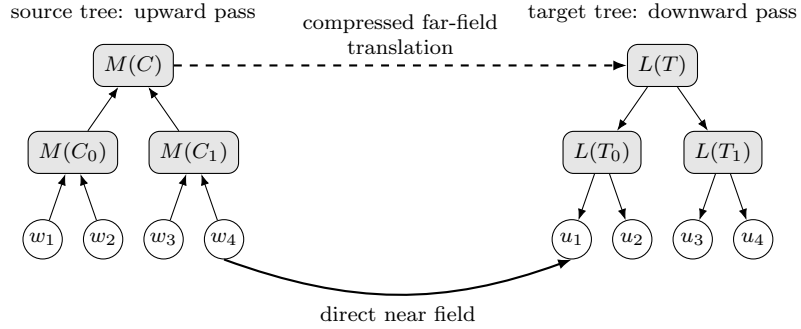


Figure 2.11: Tree-structured FMM computation. Source weights are aggregated upward into cluster moments. Well-separated clusters communicate through compressed far-field translations, which are propagated downward through the target tree. Nearby leaf interactions are evaluated directly.

There is also an important difference. Classical FMM starts from a known kernel and uses analytic expansions with error bounds. Attention does not have a fixed kernel of this kind. The scores depend on learned query and key vectors, and the useful approximation is judged by downstream loss. FMA therefore learns the far field summaries instead of deriving them from a prescribed formula.

2.3.2 Hierarchical Matrices and Low Rank Off Diagonal Structure

Hierarchical matrices give a matrix view of the same idea. An $\mathcal{H}$ matrix A partitions a dense $n \times n$ matrix into blocks indexed by a tree [10, 47, 48]. Near diagonal blocks are stored explicitly because nearby interactions can be detailed and irregular. Many off diagonal blocks are stored in low rank form because distant interactions are smoother or more compressible.

The construction uses a cluster tree and an admissibility condition. The cluster tree recursively partitions the index set $\{1, \dots, n\}$ into nested clusters. The admissibility condition decides whether a block $A_{t,s}$ between target cluster t and source cluster s is treated as low rank. A common geometric condition is

$$\max\{\text{diam}(t), \text{diam}(s)\} \leq \eta \cdot \text{dist}(t, s), \quad (2.63)$$

where $\eta > 0$. To emphasize the connection with attention, denote the target-side and source-side low-rank factors by $Q_{t,s}$ and $K_{t,s}$, respectively. If the block is admissible, it is represented as

$$\begin{aligned} A_{t,s} &\approx Q_{t,s} K_{t,s}^\top, \\ Q_{t,s} &\in \mathbb{R}^{|t| \times k}, \quad K_{t,s} \in \mathbb{R}^{|s| \times k}. \end{aligned} \quad (2.64)$$

with rank k much smaller than $|t|$ and $|s|$. If the block is not admissible, the algorithm refines it by checking the children of t and s . The storage and matrix-vector multiplication costs are then $\mathcal{O}(nk \log n)$ for standard $\mathcal{H}$ matrices under common rank assumptions.

The multiplication formula explains why this matters for attention. Let $V_s \in \mathbb{R}^{|s| \times d_v}$ contain the value vectors supported on source cluster s . Then

$$A_{t,s} V_s \approx Q_{t,s} (K_{t,s}^\top V_s). \quad (2.65)$$

The product $K_{t,s}^\top V_s \in \mathbb{R}^{k \times d_v}$ is a compressed key–value summary of the source cluster. The factor $Q_{t,s}$ maps that summary to the target rows. This is the same algebraic ordering used by low-rank attention: first aggregate values against a small key basis, then evaluate the resulting summaries against queries. In a classical $\mathcal{H}$ matrix, $Q_{t,s}$ and $K_{t,s}$ are generic low-rank bases rather than the learned query and key projections of a Transformer. FMA follows the same pattern with learned far-field key and value summaries. The block is never expanded into a dense set of fine interactions.

An $\mathcal{H}^2$ matrix adds nested bases. It replaces the block-dependent bases $Q_{t,s}$ and $K_{t,s}$ by cluster bases Q_t and K_s , together with a small coupling matrix $S_{t,s}$. If $t_1, \dots, t_a$ are the children of target cluster t and $s_1, \dots, s_b$ are the children of source cluster s , the block factorization and nestedness relations can be written as

$$\begin{aligned} A_{t,s} &\approx Q_t S_{t,s} K_s^\top, \\ Q_t &= \begin{bmatrix} Q_{t_1} E_{t_1}^Q \\ \vdots \\ Q_{t_a} E_{t_a}^Q \end{bmatrix}, \quad K_s = \begin{bmatrix} K_{s_1} E_{s_1}^K \\ \vdots \\ K_{s_b} E_{s_b}^K \end{bmatrix}. \end{aligned} \quad (2.66)$$

Here $E_{t_i}^Q$ and $E_{s_j}^K$ are small transfer matrices that express the parent basis through its child bases. This is the matrix analogue of reusing multipole summaries across levels. It reduces storage because different far field blocks share basis information. FMA does not implement a classical $\mathcal{H}^2$ matrix, but it uses the same reuse principle. The summaries live on a tree and are shared by many queries.

The limitation also carries over. Hierarchical matrix structure is useful only when far field interactions are compressible. A small rank cannot represent an arbitrary dense block. In FMA this becomes an empirical assumption about attention. The rank p controls the capacity of each far field cell, and the experiments in Chapter 3 test whether that capacity is enough for language and vision tasks.

2.3.3 Tree Structures and Geometric Decomposition

The hierarchy must match the geometry of the input. A one dimensional sequence uses intervals. A two dimensional image grid uses squares. These are not only implementation details. They determine which positions are considered near and which groups may be summarized together. Figure 2.12 illustrates the two corresponding regular partitions.

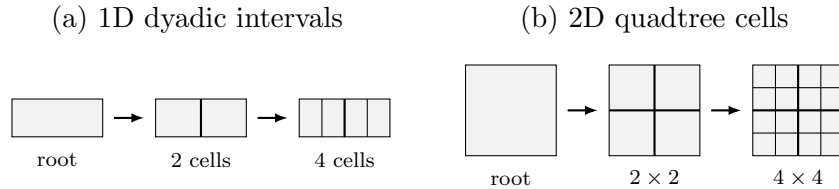


Figure 2.12: Regular geometric partitions for one- and two dimensional data. A one dimensional interval is bisected into dyadic child intervals. A two dimensional square is divided into four children at each quadtree refinement. In both cases, every coarse cell is the union of its children.

A dyadic partition of a sequence is built by recursively merging neighbouring intervals. Let n_0 be the number of base cells and let $L = \lceil \log_2 n_0 \rceil$. The number of cells in the full tree satisfies

$$N_{\text{tree}} = \sum_{\ell=0}^L \left\lceil \frac{n_0}{2^\ell} \right\rceil = \mathcal{O}(n_0). \quad (2.67)$$

Thus a bounded amount of aggregation work per cell has linear total cost. If each fine query reads summaries from all L levels, the interaction cost is $\mathcal{O}(n_0 \log n_0)$. If queries are

also coarsened with the tree, the sum of query counts across levels is linear. This is the basis for the loglinear and linear variants in Chapter 3.

The one dimensional FMA construction uses a regular dyadic hierarchy over the token sequence. With base cell size r , level ℓ groups have length $r2^{\ell-1}$. The near field is a small window of base cells around the query. The far field is assigned to coarser shells. This gives full resolution interactions near the diagonal of the attention matrix and progressively coarser blocks away from it.

A quadtree gives the corresponding two dimensional hierarchy. For an $H \times W$ grid and base side length r , the number of cells over all levels is

$$N_{\text{quad}} = \sum_{\ell \geq 0} \mathcal{O}\left(\frac{HW}{r^{2 \cdot 4^\ell}}\right) = \mathcal{O}\left(\frac{HW}{r^2}\right). \quad (2.68)$$

The two dimensional FMA construction uses this partition on image feature maps. A patch attends exactly within a local 3×3 base cell neighbourhood. Distant image regions are represented by coarser square cells. This keeps horizontal and vertical geometry intact, unlike a flattened sequence partition.

Classical particle methods often use adaptive trees, where dense spatial regions are subdivided more deeply. FMA uses regular trees for language and image experiments. The reason is pragmatic. Regular sequences and image grids lead to predictable tensor shapes and efficient batched execution on GPUs. The price is that the geometry does not adapt to each input. Content adaptivity is left to the learned summaries and attention weights.

2.3.4 What Carries Over to Fast Multipole Attention

The numerical methods above provide the design language for Chapter 3. The correspondence is concrete.

- The near field in FMM becomes exact local attention. Tokens or patches in neighbouring base cells are compared at full resolution.
- The far field in FMM becomes attention to learned cell summaries. A distant group contributes a small number of key and value representatives rather than all individual positions.
- The expansion order or off diagonal rank becomes the FMA rank p . Increasing p gives each far field cell more capacity and increases cost.

- The admissibility threshold becomes the base cell size and shell rule. Increasing r widens the exact region and reduces approximation near the query, but it increases local attention cost.
- The cluster tree becomes a dyadic tree for sequences and a quadtree for images. These trees explain why the number of summaries is linear in input size and why reading all levels gives a loglinear interaction count.

The analogy should not be overstated. Classical FMM and $\mathcal{H}$ matrices approximate a known operator. They can often specify an error tolerance before the computation begins. FMA learns an operator inside a neural network. The query, key, and value vectors change during training, and the quality of the approximation is measured by validation loss and downstream accuracy. This is why Chapter 3 includes both complexity analysis and empirical ablation studies.

Multigrid is less central for this thesis project. It also uses a hierarchy of coarse and fine representations, but its purpose is to solve discretized equations by correcting residual errors across grids. FMA does not solve a linear system and does not use a V cycle. The relevant numerical lineage is therefore fast summation and hierarchical matrix compression, not multigrid solvers. The shared multilevel intuition is only that local detail and global context should not be represented at one uniform resolution.

This section therefore motivates the structure used later. FMA is a learned near field plus far field decomposition of self attention. It keeps local interactions exact, compresses distant interactions through trainable summaries, and organizes the summaries on a tree so that global context remains available without forming the full dense attention matrix.

2.4 Symbolic Regression

This section reviews symbolic regression, the task of recovering closed form mathematical expressions from numerical observations. The discussion covers problem formulation and standard benchmarks, classical search based systems, recent learning based approaches built around Transformers, large language models, and reinforcement learning, decomposition and template based methods, and Monte Carlo tree search. The material provides background for both parts of Chapter 4.

2.4.1 Problem Formulation and Evaluation

A symbolic regression task is specified by a dataset $\{(x_i, y_i)\}_{i=1}^N$ with $x_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$, an operator library Ω , and a complexity budget Λ . The goal is to recover a closed form expression g in the symbolic class $\mathcal{G}_\Lambda$ generated by Ω such that $g(x_i) \approx y_i$. The class $\mathcal{G}_\Lambda$ is finite for each fixed Λ , but its size grows exponentially with the operator vocabulary and the complexity budget. So the search problem is combinatorial.

A common formulation is to minimize the empirical loss with a complexity penalty,

$$\hat{g} \in \arg \min_{g \in \mathcal{G}_\Lambda} \frac{1}{N} \sum_{i=1}^N (y_i - g(x_i))^2 + \lambda C(g), \quad (2.69)$$

where $C(g)$ is a complexity measure, often the number of nodes in the expression tree, and $\lambda > 0$ controls the tradeoff. Free constants inside g are fitted by a continuous optimizer. Objectives of the form in Equation (2.69) underlie many modern symbolic regression systems, although implementations often return a Pareto front or hall of fame rather than a single minimizer [18, 21, 29, 162].

Two expressions g and f are *algebraically equivalent*, written $g \equiv f$, if one can be transformed into the other by valid algebraic identities under the assumed domain, after fitting any free constants. For example,

$$(x + 1)^2 \equiv x^2 + 2x + 1, \quad (2.70)$$

whereas a polynomial approximation to $\sin(x)$ is not algebraically equivalent to $\sin(x)$ even if their numerical values are close on the observed interval. A computer algebra system such as SymPy can test many such cases by simplifying $g - f$ to zero or by mapping both expressions to the same canonical representative. This process is called *canonicalization*.

Two evaluation criteria are used in the literature. The first is numerical fit. It is measured by mean squared error, root mean squared error, normalized mean squared error, or coefficient of determination. The second criterion is exact recovery, which is closer to the scientific motivation. Under exact recovery, the recovered expression must be algebraically equivalent to the ground truth equation up to numerical constants. Formally, given a ground truth f^* , exact recovery requires

$$g \equiv f^* \quad \text{under algebraic simplification, after constant fitting,} \quad (2.71)$$

Exact recovery is more demanding than numerical fit because two expressions can fit the data equally well while differing structurally. For example, $\sin(x_1 + x_2)$ and a high order

polynomial approximation may have similar numerical fit, but only the former is the correct closed form.

SRBench [21] is a standard benchmark suite for symbolic regression. It contains more than one hundred tasks drawn from physics, biology, and synthetic generators and reports symbolic solution rate separately from predictive accuracy. The AI Feynman dataset [154] collects equations from the Feynman Lectures on Physics and has become a common benchmark for exact recovery. Its original input ranges are relatively narrow and are specified independently for each variable.

SRSD-Feynman revises this evaluation for scientific discovery [103]. It contains 120 physics inspired equations with known ground truth and assigns equation specific ranges based on the physical interpretation and SI unit of each variable. Variables whose magnitudes span several orders are sampled on a logarithmic scale. These choices make it harder for a locally accurate approximation to receive the same credit as the generating equation. The benchmark therefore reports predictive accuracy and symbolic recovery as different outcomes. Section 4.1 uses SRSD-Feynman as its primary recovery benchmark and supplements it with six standard cross-benchmark datasets. More recent resources include the causal chambers testbed of Gamella et al. [40] and the LLM-SRBench suite [143].

The operator library changes the difficulty and the meaning of a benchmark. A polynomial library leads to a different problem from a grammar that includes trigonometric functions, exponentials, logarithms, and division. The sampling distribution for x_i also matters. Two expressions can be hard to distinguish on a narrow domain and easy to distinguish on a wider one. Noise, missing values, and free constants introduce further ambiguity. A symbolic regression result should therefore be interpreted together with its grammar, data distribution, and recovery criterion.

Candidate expressions are commonly represented as rooted syntax trees. Each internal node is an operator, and each leaf is an input variable or a constant. A subtree therefore represents a valid subexpression, while the root represents the complete expression. Tree size, depth, or a weighted count of its nodes can be used as the complexity measure $C(g)$.

There is also a distinction between expression discovery and parameter estimation. Once a candidate tree has been chosen, its free constants can often be fitted by continuous optimization. The harder part is choosing the discrete structure of the tree. Changing one operator or moving one subtree produces a different nonlinear optimization problem. Most methods reviewed below can be understood as different strategies for exploring this discrete structure space.

2.4.2 Classical Search-Based Symbolic Regression

The earliest symbolic regression systems used genetic programming [75]. A population of syntax trees is initialized at random or from a small set of motifs. *Crossover* selects a compatible subtree from each of two parent expressions and exchanges the selected subtrees. For example, consider $g_1(x) = x_1 + \sin(x_2)$ and $g_2(x) = x_1x_2$. Exchanging the subtrees $\sin(x_2)$ and x_2 produces the offspring $x_1 + x_2$ and $x_1\sin(x_2)$. *Mutation* modifies a single parent by replacing an operator, leaf, or subtree with another grammar-valid choice. For example, replacing the root operator in $x_1 + x_2$ from addition to multiplication gives x_1x_2 , while replacing the leaf x_2 by the subtree $\cos(x_2)$ gives $x_1 + \cos(x_2)$. Repeated crossover and mutation produce new generations. Each generation is filtered by a fitness function that combines accuracy and a complexity penalty. After many generations, the population concentrates on accurate and simple expressions. Eureqa [136] is an early system in this lineage and demonstrated symbolic regression on physical and biological data.

Some scientific discovery methods use a more restricted candidate space. The SINDy method [17] builds a dictionary of nonlinear terms from measured state variables and finds a sparse coefficient vector for the governing dynamics. SISSO [113] constructs a large space of candidate descriptors and combines screening with sparse optimization. These methods can be effective when the dictionary reflects domain knowledge. Their advantage is that continuous sparse regression replaces an open ended tree search. Their limitation is that recovery is constrained by the terms made available in advance.

Modern genetic programming systems remain strong baselines on symbolic regression benchmarks. PySR [29] combines multi-population evolutionary search with an evolve–simplify–optimize loop and multi objective selection on the Pareto front of accuracy and complexity. It also supports user defined operator libraries and constraints. Operon [73] uses a compact tree representation and emphasizes high performance evaluation and robust constant optimization. GP-GOMEA [162] learns dependencies between tree positions and uses them to guide structural variation. These systems can revisit both structure and constants after observing the data, but they usually perform the expensive structure discovery again for every new dataset. PySR and Operon output a hall of fame, a ranked list of accuracy–complexity tradeoffs from which the user selects a final expression.

The hall of fame is important because fit and simplicity rarely identify a single uncontested expression. A short equation may underfit. A longer equation may improve training error through incidental structure. Several algebraically related forms may appear with different tree sizes before simplification. Returning a Pareto front leaves room for a later selection rule. Chapter 4 uses this interface and asks whether environment stability can resolve cases that pooled fit leaves ambiguous.

The strengths of classical search based methods are flexibility in the operator library, transparency of the search process, and strong empirical performance across a wide range of tasks. Their weaknesses include sensitivity to the chosen fitness and complexity objectives and the need for specialized mechanisms to inject structural prior knowledge [72, 76], as well as redundancy caused by syntactically different expressions that represent the same or nearly the same function. The last issue is not an absolute absence of equivalence mechanisms. PySR simplifies expressions during search, while semantic genetic programming uses output behaviour to guide variation and identify semantically related subtrees [29, 156, 164]. Nevertheless, canonical-equivalence analyses show that standard genetic programming can repeatedly evaluate expressions congruent to ones it has already visited [77]. Sections 4.1 and 4.2 address other aspects of these weaknesses in complementary ways. Section 4.1 adds learned structural priors to guide the search, while Section 4.2 adds a residual invariance criterion that selects among the search outputs.

2.4.3 Transformer Based Symbolic Regression

A more recent line of work treats symbolic regression as a supervised mapping from numerical observations to symbolic expressions. The Transformer architecture has been the most common backbone for this approach. These methods learn from many synthetic equations before they are evaluated on a new problem. This differs from classical genetic programming, which usually starts a new search from scratch for each dataset.

A typical training example contains a symbolic expression and one or more numerical views of that expression. A numerical view is a set

$$\mathcal{D}_f = \{(x_i, f(x_i))\}_{i=1}^N. \quad (2.72)$$

The input points are sampled from a chosen domain. The symbolic expression is converted to a sequence, commonly in prefix notation. For example, $x_1 + \sin(x_2)$ is serialized as [ADD, X_1, SIN, X_2]. This representation is also called *Polish notation*. It is obtained by a preorder traversal of the expression tree: visit the root operator first, then traverse its child subtrees from left to right. The binary token ADD therefore consumes the following two subexpressions, while the unary token SIN consumes the following one. Because operator arities are known, this prefix representation can be parsed without parentheses. It is especially convenient for autoregressive equation generation because the decoder can track how many operand slots remain while producing tokens from left to right.

The numerical input is a set rather than an ordered sentence. Transformer symbolic regression systems therefore encode the observations without assigning meaning to their

row order. End-to-end Symbolic Regression (E2E) [63] first represents each scalar in base-10 floating point notation by three discrete tokens for its sign, mantissa, and exponent. Let $z_i = [x_i^\top, y_i]^\top$ denote one input-output observation, let $\tau(z_i)$ collect the floating point tokens of all its coordinates, and let Emb embed and concatenate those tokens. A two-layer feedforward point embedder produces

$$e_i = \text{FFN}_{\text{pt}}(\text{Emb}(\tau(z_i))) \in \mathbb{R}^{d_{\text{emb}}}. \quad (2.73)$$

The point embeddings are then processed jointly by a numerical Transformer encoder,

$$H_{\mathcal{D}} = \text{Enc}_{\text{num}}(e_1, \dots, e_N) \in \mathbb{R}^{N \times d_{\text{emb}}}. \quad (2.74)$$

No positional embeddings are added to the numerical encoder. Consequently, if P_π is the permutation matrix associated with a reordering π of the observations, then

$$H_{\pi(\mathcal{D})} = P_\pi H_{\mathcal{D}}. \quad (2.75)$$

Thus the encoder states are permutation equivariant rather than collapsed immediately to one vector. In E2E, the expression decoder cross-attends to all N rows of $H_{\mathcal{D}}$.

SNIP [141] uses a related point embedder and position-free Transformer encoder, but distills the encoder states into a single dataset-level vector by learned attention pooling,

$$a_{\mathcal{D}} = \text{softmax}(H_{\mathcal{D}} w_a), \quad z_{\mathcal{D}} = H_{\mathcal{D}}^\top a_{\mathcal{D}}, \quad (2.76)$$

where $w_a \in \mathbb{R}^{d_{\text{emb}}}$ is learned and $a_{\mathcal{D}} \in \mathbb{R}^N$ assigns a weight to each observation. For symbolic regression, SNIP maps $z_{\mathcal{D}}$ through a learned network g_γ to a sequence of decoder memory vectors. The memory supplied to the expression decoder can therefore be summarized as

$$M_{\mathcal{D}} = \begin{cases} H_{\mathcal{D}}, & \text{E2E,} \\ g_\gamma(z_{\mathcal{D}}) \in \mathbb{R}^{L \times d_{\text{emb}}}, & \text{SNIP-SR.} \end{cases} \quad (2.77)$$

NeSymReS [13] provides another variant: its Set Transformer uses induced set attention blocks to process the observations and pooling by multihead attention to produce a fixed number of abstract features for the decoder.

A Transformer decoder generates a symbolic token sequence $s_{1:T}$ by cross-attending to $M_{\mathcal{D}}$. Its probability factorizes as

$$p_\theta(s_{1:T} \mid \mathcal{D}) = \prod_{t=1}^T p_\theta(s_t \mid s_{<t}, M_{\mathcal{D}}). \quad (2.78)$$

Training minimizes the negative log likelihood of target expression tokens. Beam search can retain several likely expressions. Numerical optimization is often used afterward to fit free constants.

Permutation invariance is a central design requirement for the numerical encoder. Deep Sets [185] characterizes a broad class of invariant set functions using elementwise features followed by aggregation. Set Transformer [85] uses attention to model interactions among set elements while preserving invariance at the set output. Symbolic regression models use related ideas because reordering sampled points should not change the predicted equation. At the same time, the encoder must represent the geometric relation between each input and its output.

Lample and Charton [82] use Transformers for symbolic mathematics tasks such as integration and differential equation solving. Their input and output are both symbolic token sequences. NeSymReS [13] adapts large scale pretraining to symbolic regression. It generates synthetic equations procedurally and trains a Transformer to predict a symbolic equation from sampled input-output pairs. At inference time, the model produces candidate expressions for a new numerical dataset.

Kamienny et al. [63] develop an end to end Transformer approach for symbolic regression. Their model is trained on generated expressions and predicts operators, variables, and numerical constants jointly from numerical observations. The predicted constants can subsequently initialize a BFGS refinement step. ODEFormer [33] extends the sequence generation viewpoint to ordinary differential equations. It recovers symbolic differential equations from irregular and noisy trajectories rather than from independent evaluations of a static function.

SNIP [141] introduces symbolic-numeric contrastive pretraining. A symbolic encoder and a numerical encoder process two views of the same equation. A contrastive objective aligns their latent representations. The pretrained encoders are then transferred to downstream tasks, including symbolic regression. This is a multimodal pretraining strategy in the sense of Section 2.1.6. The modalities are symbolic expressions and numerical observations rather than images and captions.

Learned symbolic regression systems also differ in where candidate optimization occurs. NeSymReS and end to end Transformer methods decode in a discrete symbolic sequence space. SNIP-SR searches a learned continuous latent space before decoding and refitting constants. GenSR [89] constructs a continuous equation-generative space with a conditional variational autoencoder and uses CMA-ES to search that space. These are direct generation methods in the sense that the learned output space supplies the candidate equation; they do not explicitly repair the decoded expression by editing its syntax tree. This distinction

motivates hybrids in which a neural model proposes a promising region but a separate symbolic search constructs and revises the final equation.

MDLformer [182] uses a different learned signal. It estimates a minimum description length objective for observed data. SR4MDL then uses this estimate to guide symbolic search. The purpose is to replace a search objective based only on prediction error with one that better reflects progress toward the symbolic form of the target equation. This is an example in which a Transformer supplies guidance to a search procedure rather than directly decoding the final expression.

Synthetic data generation is part of the model design. The grammar determines which operators, depths, constants, and variable counts the model sees during pretraining. The numerical sampler determines which regions of each function are observed. A broad generator improves coverage but also makes learning harder. A narrow generator improves specialization but increases the risk of failure on equations outside the training distribution. Search based hybrids are useful partly because they can verify and refine a neural proposal after pretraining.

Transformer based methods have two clear strengths. Pretraining lets them reuse patterns learned from many synthetic equations. Inference can also produce useful candidates quickly. They also have limitations. Performance can degrade when test equations fall outside the synthetic training distribution. Exact recovery remains difficult because small token errors can change the mathematical meaning of an expression. A direct decoder must commit to one symbolic sequence even when several intermediate structures are plausible. Hybrid methods address this limitation by using neural predictions to guide a separate search procedure.

2.4.4 Other Neural Symbolic Regression Methods

Neural symbolic regression also includes reinforcement learning, large language model, and diffusion based methods. These methods differ in how they generate candidates and how they use feedback from numerical fit.

Reinforcement learning based methods cast expression construction as a sequential decision problem. At each step, a policy selects an operator, variable, or constant. A completed expression receives a reward based on fit and possibly complexity. Deep Symbolic Regression [119] trains a recurrent policy with a risk seeking policy gradient objective. The objective places more weight on the best sampled expressions than on average reward. This is appropriate for symbolic regression because the goal is to discover a small number of strong expressions rather than to sample many moderately accurate ones.

Sym-Q [151] uses offline reinforcement learning. It represents the numerical point set and the current partial expression tree, then predicts values for possible symbolic actions. The method supports an interactive design loop in which a user can modify parts of the expression tree. Reinforcement learning methods can incorporate a task specific reward directly. Their difficulty is that the discrete action space is large and useful rewards may be sparse.

Large language model based symbolic regression uses a general language model to generate or refine equations through prompts. LLM-SR [142] formulates equation discovery as program synthesis. The language model produces executable expressions, receives feedback from their numerical behavior, and iteratively improves the candidate program. LaSR [41] develops a learned concept library that can be reused across tasks. Large language models can use textual context and broad prior knowledge. They are also expensive to query repeatedly, and their generated programs still require numerical evaluation and verification. LLM-SRBench [143] provides a benchmark for controlled evaluation of this family.

Diffusion based generation provides another alternative to left-to-right decoding. A recent method applies discrete token diffusion to symbolic regression [153]. Training corrupts expression tokens and learns a reverse denoising process. At inference time, the model refines a noisy token sequence into a candidate equation. Unlike autoregressive decoding, multiple token positions can be revised during the denoising process. This may be useful when the correct choice at one position depends strongly on later parts of the expression. Diffusion based symbolic regression is recent, and its relative performance across broader benchmark suites is still being established.

These methods address different weaknesses of flat symbolic search. Reinforcement learning optimizes candidate construction for a reward. Large language models bring broad pretrained knowledge and program synthesis ability. Diffusion revises a sequence iteratively instead of committing to tokens from left to right. The project of Chapter 4 is closer to the Transformer based hybrid methods. It uses learned signals to guide symbolic search and retains explicit constant fitting and verification.

Candidate generation is only one part of the computation. Every proposed expression must be parsed, checked for validity, evaluated on numerical points, and often simplified. Expressions with division, logarithms, or square roots may be undefined on part of the sampled domain. Free constants may require an inner optimization loop. A method that proposes better candidates can improve runtime even when the evaluator remains unchanged, because fewer invalid or unpromising structures reach the expensive fitting stage.

2.4.5 Templates, Sketches, and Decomposition

A complementary line of work imposes structural priors on the search through templates, sketches, or decomposition steps. These methods do not attempt to decode an expression in full. Instead, they restrict the search to a structured subspace of expressions and rely on numerical methods to complete the missing details.

AI Feynman [154] and its successor AI Feynman 2.0 [155] formalize this idea for physics equations. The system attempts a sequence of structural simplifications. These include dimensional analysis, separability checks, symmetry detection, and translation invariance tests. It then recursively applies symbolic regression to the simpler subproblems that result. The decomposition strategy is what allows AI Feynman to recover formulas that are out of reach for a flat search over the full expression space.

Transformer based skeleton methods provide a learned version of this idea, although their use of the word *skeleton* is usually narrower than the typed sketches defined below. NeSymReS [13] replaces every numerical constant by a placeholder token, uses a Set Transformer encoder and a Transformer decoder to predict the resulting expression skeleton, and then fits the missing constants with BFGS. SymbolicGPT [157] similarly uses a GPT decoder to generate an equation containing $\langle C \rangle$ placeholders and optimizes those constants as a postprocessing step. In these methods the operators and variables are decoded explicitly; only the numerical constants remain as holes.

A more decomposition-oriented example is the Multi-Set Transformer of Morales and Sheppard [108]. It constructs several datasets in which one input variable varies while the others are fixed, and predicts a univariate symbolic skeleton describing the effect of that variable. The method therefore uses a Transformer to infer partial structures that can later be combined into a multivariate explanation.

The notion of a typed sketch generalizes this idea. A sketch S is an expression tree with one or more typed holes that must be filled with subexpressions of a specified type. The completion set of S is

$$\mathcal{C}(S) = \left\{ g \in \mathcal{G}_\Lambda \mid \begin{array}{l} g \text{ is obtained by filling all holes of } S \\ \text{with type compatible subexpressions} \end{array} \right\}. \quad (2.79)$$

Sketch-based symbolic regression searches over sketches and fills the holes either by a smaller inner search or by direct prediction. The motif vocabulary used in Chapter 4 is a related construction. It is a finite library of small reusable subexpressions. Their probabilities are predicted by the neural model and used as priors for the search.

The strengths of template, sketch, and decomposition methods are the dramatic reduction in the effective search space and the interpretability of intermediate steps. The weaknesses are sensitivity to the choice of template library and the difficulty of designing decomposition tests that are accurate enough to be useful but not so strict that they reject the true equation. The project of Chapter 4 addresses the first weakness by learning the template and motif priors from synthetic data. It addresses the second by treating the priors as soft signals to guide a Monte Carlo tree search rather than as hard partitions of the search space.

Structural priors can be hard or soft. A hard grammar rule rejects an expression that violates type, arity, or dimensional constraints. This is useful because such an expression cannot be the target under the assumed scientific setting. A soft prior assigns a higher probability to a plausible operator, motif, or sketch but still allows alternatives to be explored. Hard rules improve efficiency when they are reliable. Soft priors are safer when the learned signal is uncertain. The method in Chapter 4 uses both roles in different parts of the search.

2.4.6 Monte Carlo Tree Search for Symbolic Regression

Monte Carlo tree search (MCTS) is a family of best first search algorithms that interleaves four operations. It selects a path through the current search tree, expands a new leaf, evaluates the leaf by simulation or by a learned value function, and backpropagates the evaluation through the path. Upper Confidence Bounds applied to Trees (UCT) [70] is the canonical algorithm and balances exploration and exploitation with an upper confidence bound. Predictor-UCT (PUCT) [131] is used in AlphaZero [144] and replaces the uniform exploration term with a prior probability distribution over actions supplied by a learned policy. The survey of Browne et al. [16] reviews the broader landscape.

In symbolic regression, MCTS treats expression construction as a sequential decision problem. The root of the tree is an empty or partial expression. Each action either appends an operator or terminal to the expression or fills a hole in a sketch. The leaf evaluation combines numerical fit of the completed expression with a complexity penalty.

The PUCT variant is particularly attractive. Let $N(s, a)$ be the visit count of action a at state s , $Q(s, a)$ the average value of completed trajectories that passed through (s, a) , and $P(a \mid s)$ the prior probability of a at s supplied by a learned policy. The PUCT selection rule chooses

$$a^* = \arg \max_a \left[Q(s, a) + c_{\text{puct}} P(a \mid s) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \right], \quad (2.80)$$

where $c_{\text{puct}} > 0$ controls the strength of exploration. A high $P(a \mid s)$ encourages search effort toward actions the policy considers plausible, while the visit count term prevents the search from collapsing on a single branch. In symbolic regression, the prior $P(a \mid s)$ can be supplied by a neural model that predicts plausible operators, lengths, sketches, or motifs from the numerical data.

Transformer based Planning for Symbolic Regression [139] and the neural symbolic regression with MCTS of Kamienny et al. [64] are recent realizations of this idea. They combine a Transformer encoder of the numerical data with a PUCT search over expression tokens. The work of Chapter 4 follows the same broad pattern. It contributes structural priors at multiple levels (operator multilabel, length bucket, typed sketch, motif probabilities). It also adds an interaction with local symbolic edits and constant fitting that targets exact recovery rather than numerical fit.

Other neural-guided systems insert learned information at different stages of explicit search. NGGP uses a recurrent policy to seed genetic programming populations [109]. uDSR combines recurrent generation, pretraining, and evolutionary search [83]. RSRM uses Double Q-learning to prune the actions considered by MCTS while an expression tree is constructed [177]. In all three cases, the learned model prioritizes the search, while explicit structural exploration remains responsible for the final equation. This differs from optimizing only a decoded sequence or a continuous latent representation.

Constant fitting and symbolic verification are two practical ingredients of any MCTS based symbolic regression system. Constant fitting is usually performed by a continuous optimizer such as BFGS or L-BFGS, called after the symbolic structure has been chosen. Symbolic verification, often implemented through SymPy, checks whether a candidate expression satisfies the algebraic equivalence criterion defined in Section 2.4.1. This is what makes exact recovery a meaningful criterion. Both ingredients are standard in the literature.

Symbolic search trees contain redundancy. Commutative operators can generate equivalent expressions in different orders. Algebraic identities can produce distinct trees for the same function. Canonicalization, memoization, and duplicate detection prevent repeated work. Batching can also improve throughput because many candidate expressions are evaluated on the same numerical dataset. These implementation choices do not change the MCTS rule, but they strongly affect how much useful search can be performed under a fixed runtime budget.

2.5 Invariance and Causal Inference

This section reviews invariance principles in causal inference and the statistical machinery used to test for distributional invariance. The material is the background for the residual invariance selector of Chapter 4.

2.5.1 Structural Causal Models and Interventions

A *structural causal model* (SCM) specifies variables and *structural assignments* that describe how each variable is generated from its direct causes and exogenous noise [114, 118]. For variables $X = (X_1, \dots, X_p)$ and a target Y , we write

$$X_j = f_j(X_{\text{pa}(j)}, U_j), \quad j = 1, \dots, p, \quad (2.81)$$

$$Y = f_Y(X_{\text{pa}(Y)}, U_Y), \quad (2.82)$$

where $\text{pa}(j)$ denotes the *parents* of X_j in a *directed acyclic graph* (DAG), and $U_1, \dots, U_p, U_Y$ are independent *exogenous noise variables*. A *hard intervention* $\text{do}(X_j = x)$ replaces the assignment for X_j by $X_j := x$ while leaving the other assignments unchanged.

The terminology of parents, *children*, *descendants*, and *ancestors* carries over from the graphical model literature. A variable X_k is a parent of Y if there is a direct edge from X_k to Y in the DAG. The set of parents $\text{pa}(Y)$ contains the variables that directly determine Y through f_Y . A descendant of Y is a variable X_j that has Y on a directed path to it. Descendants are measurements or other variables generated downstream of Y and are not part of its causal mechanism.

The distinction between parents and descendants is central to the failure mode that motivates Chapter 4. A descendant of Y can be highly correlated with Y . It is therefore a useful predictor under pooled empirical risk. But it does not represent the mechanism that generates Y . Standard symbolic regression has no notion of causal direction. It may select a descendant based expression whenever it fits the data well.

An *environment* $e \in \{1, \dots, E\}$ is a data collection regime, such as an intervention, instrument, site, batch, time period, or operating condition. RIMS assumes that the mechanism for Y and its exogenous noise law are unchanged across the environments used for selection. For example, consider

$$X = U_X, \quad Y = 2X + U_Y. \quad (2.83)$$

Environment $e = 1$ may be observational, while $e = 2$ applies $\text{do}(X = 3)$. The distribution of the data changes, but the assignment for Y remains $Y = 2X + U_Y$, so both environments are suitable for selection. An environment $e = 3$ with $\text{do}(Y = 5)$ replaces the target mechanism and is not suitable. The invariance principle exploited in Chapter 4 is that the true mechanism for Y produces residuals with the same distribution across suitable environments, even when the surrounding data distribution changes.

The value of interventional data is that it can break associations that are indistinguishable in one pooled distribution. In a single environment, a parent, a descendant, and a correlated proxy may all predict Y well. Across environments, interventions can change the distribution of these variables while preserving the assignment for Y . The structural assignment then acts as the stable object. This autonomy of mechanisms is the reason invariance can provide information beyond predictive fit.

Not every distribution shift is useful. If a shift leaves the relationship between a shortcut and Y unchanged, the shortcut may remain stable. If the mechanism for Y itself changes, the original equation need not remain invariant. The theory in Chapter 4 states conditions under which the available environments create a detectable gap between the true mechanism and competing expressions.

2.5.2 Invariance Principles in Causal Learning

Invariant Causal Prediction (ICP) [117] formalizes the invariance principle as a hypothesis test for the set of causal parents. For each candidate subset $S \subseteq \{1, \dots, p\}$ of predictors, ICP tests whether the conditional distribution $P(Y \mid X_S = x_S)$ is invariant across environments. The plausible causal parent set is the intersection of all subsets that pass the test at a chosen significance level. Under suitable assumptions, this set is a subset of the true parents of Y with high probability. ICP operates on variable sets rather than on full functional forms.

Invariant Risk Minimization (IRM) [4] extends the invariance idea to representation learning. For a predictor h and loss function ℓ , the *environment risk* is

$$R^e(h) = \mathbb{E}_{(X,Y) \sim P^e} [\ell(h(X), Y)], \quad (2.84)$$

where P^e is the joint distribution in environment e . For squared-error regression, this becomes $R^e(h) = \mathbb{E}_{P^e} [(h(X) - Y)^2]$. The IRM objective seeks a representation $\Phi(X)$ such that the optimal predictor on top of Φ is the same across all training environments $\mathcal{E}_{\text{tr}}$,

$$\min_{\Phi, w} \sum_{e \in \mathcal{E}_{\text{tr}}} R^e(w \circ \Phi) \quad \text{subject to} \quad w \in \arg \min_{\bar{w}} R^e(\bar{w} \circ \Phi) \quad \text{for all } e \in \mathcal{E}_{\text{tr}}, \quad (2.85)$$

The bi-level constraint is intractable in general, so the practical IRMv1 objective replaces it by a gradient penalty on a fixed dummy classifier. The framework has motivated a large body of follow up work, including risk extrapolation [78] and group distributionally robust optimization [135]. These methods produce black box predictors that are robust under environment shift but do not produce interpretable equations.

Anchor regression [132] interpolates between empirical risk minimization and an invariance penalty. It adds a term that penalizes the covariance between residuals and an exogenous anchor variable that codes the environment. The result is a continuous spectrum of estimators that trade off in distribution fit and out of distribution robustness. TIVA [123] learns invariant representations under general interventions on the target variable and is a recent contribution in this direction.

The selector of Chapter 4 stands in the tradition of these invariance methods but differs in its object. ICP selects variable sets. IRM and follow ups select representations. Anchor regression selects coefficients of a linear predictor. The selector of Chapter 4 selects symbolic equations from a finite candidate pool produced by an external symbolic regression system. The invariance score is computed at the level of residual distributions. This makes the selector compatible with any predictor that produces pointwise predictions, including the closed form expressions returned by symbolic regression.

The interpretation of invariance requires care. A candidate that passes an invariance test is not automatically the unique causal mechanism. Several candidates can remain stable if the environments are weak or if the candidate pool contains algebraically related expressions. Conversely, finite samples can make the residuals of the true mechanism appear different by chance. Invariance is therefore used together with fit and explicit assumptions on the environment set. The selector in Chapter 4 is a model selection rule under those assumptions, not a general causal discovery algorithm.

2.5.3 Descendant Shortcuts and the Failure of Pooled Risk

The specific failure mode that motivates the selector of Chapter 4 can be illustrated with a small example. Let $Y = f^*(X_1, X_2) + \varepsilon$ be the true mechanism. Suppose the observed variables include $X_3 = \alpha Y + \eta$, where η has small variance. The variable X_3 is a descendant of Y and is not a causal parent. Pooled empirical risk minimization over the symbolic class may prefer the expression $g(x) = x_3/\alpha$, which has very low pooled error and low complexity, over the true expression $f^*(x_1, x_2)$, which can have higher complexity and slightly higher pooled error.

The Feynman experiment in Chapter 4 creates this failure mode by adding a measurement sensor that contains a noisy copy of the target. Pooled fit selects an expression using the sensor on all 119 datasets and recovers no physical mechanisms. A leave-one-feature-out union of candidate pools restores the physical mechanism for 81 datasets, after which residual invariance recovers 73 of them at the diagnostic tolerance $\epsilon = 2.5$. The experiment therefore separates candidate availability from the final selection problem.

The invariance principle gives a principled way to address this selection problem. If the mechanism for Y and its noise law are stable across environments, then the residuals of the true expression f^* have the same distribution in every environment. For the exact downstream inverse $g(x_3) = x_3/\alpha$, the residual is $-\eta/\alpha$. It is therefore invariant when α and the noise law of η are fixed, in which case residual invariance cannot distinguish the shortcut from the mechanism. A detectable gap can arise when the measurement calibration or noise law changes, or when a fixed imperfect inverse is combined with a shift in the distribution of Y . Chapter 4 formalizes both the recovery condition and this identification boundary.

The example also identifies a boundary of the method. If α and the noise distribution remain unchanged in every observed environment, the descendant expression can have stable residuals. No residual based test can distinguish it from the mechanism using those environments alone. Useful environment design is therefore part of the problem. The environment shifts should perturb the surrounding system enough to expose shortcuts while leaving the target mechanism unchanged in the environments used for selection.

2.5.4 Two-Sample Tests and Distributional Discrepancy

The numerical implementation of the invariance principle requires a way to measure the discrepancy between residual distributions estimated from finite samples. For a candidate expression g and environment e , define

$$r_i^e(g) = y_i^e - g(x_i^e), \quad \hat{P}_e^g = \frac{1}{n_e} \sum_{i=1}^{n_e} \delta_{r_i^e(g)}, \quad (2.86)$$

where δ_r is a point mass at r . Let D be a discrepancy between empirical distributions and let A contain at least two environments. One natural average instability score is

$$\mathcal{I}_A(g) = \frac{2}{|A|(|A| - 1)} \sum_{\substack{e, e' \in A \\ e < e'}} D(\hat{P}_e^g, \hat{P}_{e'}^g). \quad (2.87)$$

A smaller value indicates more stable residual distributions. Other aggregations can replace the average when the worst environment difference is more important. Two sample statistics provide standard choices for D .

The Cramér-von Mises statistic [28, 166] measures the integrated squared difference between two empirical cumulative distribution functions. For samples $\{r_i^e\}$ and $\{r_j^{e'}\}$ with empirical CDFs $\hat{F}_e$ and $\hat{F}_{e'}$, the two sample statistic is

$$\text{CvM}(\hat{F}_e, \hat{F}_{e'}) = \frac{n_e n_{e'}}{n_e + n_{e'}} \int (\hat{F}_e(r) - \hat{F}_{e'}(r))^2 dH(r), \quad (2.88)$$

where H is the empirical CDF of the pooled sample. The statistic is nonnegative. It equals zero only when the two empirical CDFs coincide. It is bounded above for fixed sample sizes. The Anderson-Darling statistic [3] is a weighted variant that places more emphasis on the tails of the distribution.

The Kolmogorov-Smirnov statistic [71, 145] uses the supremum rather than the integrated squared difference of the CDFs and is widely used as a baseline. The Maximum Mean Discrepancy (MMD) [43] measures the distance between two distributions through the difference of their mean embeddings in a reproducing kernel Hilbert space. For a positive definite kernel k , the squared MMD between distributions P and Q is

$$\text{MMD}^2(P, Q) = \mathbb{E}_{X, X' \sim P} k(X, X') + \mathbb{E}_{Y, Y' \sim Q} k(Y, Y') - 2 \mathbb{E}_{X \sim P, Y \sim Q} k(X, Y). \quad (2.89)$$

The empirical MMD replaces expectations by sample averages. MMD is the standard choice of two sample statistic when the underlying samples are multivariate.

The selector of Chapter 4 uses a normalized Cramér-von Mises discrepancy on residual distributions. Residuals are one dimensional, so a CDF based discrepancy is natural. The normalized form depends on the complete empirical residual distributions, lies in $[0, 1]$, and is dimensionless. It can also be computed efficiently from sorted residual samples.

With more than two environments, pairwise discrepancies must be combined. An average pairwise score measures typical instability. A maximum pairwise score emphasizes the worst observed shift. One can also compare each environment with the pooled residual distribution of the others. These choices answer slightly different questions. The selector of Chapter 4 uses a uniform average over all environment pairs.

Finite sample calibration also matters. A discrepancy statistic is not itself a proof of a distributional difference. Its scale depends on sample size and on the variability of the residuals. Permutation procedures provide a direct way to estimate a null distribution when exchangeability is justified. When many candidate equations and many environment pairs are examined, the selection procedure must also account for the fact that the largest observed discrepancy can increase by chance.

2.5.5 Out-of-Distribution Generalization and Domain Shift

The invariance principle is part of a broader literature on out of distribution generalization. The general problem is to train a predictor that performs well not only on data drawn from the training distribution but also on data drawn from a related distribution that differs from the training one in some controlled way. The selector of Chapter 4 addresses a narrow version of this problem. Rather than training a predictor that generalizes, the selector chooses among already trained predictors based on an invariance criterion estimated on training environments.

There are three connections to the broader literature. First, the residual invariance criterion is conceptually similar to the IRM penalty [4] and to the anchor regression objective [132]. It is applied as a model selection rule rather than as a training objective. Second, the finite-candidate result in Chapter 4 uses uniform concentration to separate a fit-admissible mechanism class from competing candidates by a detectable residual-invariance gap. Third, the causal chambers testbed [40] provides physical experimental setups in which interventions can be implemented and illustrates the type of multi-environment data for which invariance methods are intended.

Model selection is a central difficulty in domain generalization. A method is trained on observed environments but is intended to perform well on unseen ones. The best validation rule is therefore not obvious. DomainBed [46] shows that careful experimental control and model selection can change conclusions about the value of specialized domain generalization algorithms relative to empirical risk minimization. The same concern appears in causal symbolic regression. A selector should use only information available in the training environments and should state which shifts its assumptions cover.

The phrase distribution shift covers several distinct settings. Covariate shift changes the distribution of inputs while preserving the conditional relationship of interest. Label or mechanism shift changes the target assignment itself. Group shift changes the proportions of known subpopulations. The residual invariance selector is aimed at a specific case. It uses shifts in the surrounding variables to identify an equation whose target mechanism remains stable. It is not intended to guarantee robustness under arbitrary changes to the target mechanism.

Chapter 3

Fast Multipole Attention

This chapter presents Fast Multipole Attention. It builds on the efficient attention and multilevel numerical background in Sections 2.2 and 2.3. The goal is to replace dense all pairs attention with a learned multilevel decomposition. Nearby tokens or patches interact at full resolution. Distant tokens or patches interact through compact summaries arranged on a hierarchy. The chapter develops the one dimensional construction for sequences, the two dimensional construction for images, the complexity analysis, and experiments in language modeling and vision.

3.1 Problem Setting and Design Goals

This section defines the computational problem addressed in this chapter: reducing the quadratic time and memory cost of dense self attention while preserving a global receptive field. Chapter 2 reviewed Transformer attention and several families of efficient attention methods. Here, we seek an attention operator that allows every output position to use information from the whole input in a single layer without computing and storing all pairwise interactions at full resolution.

Consider one attention head with queries, keys, and values $Q, K, V \in \mathbb{R}^{n \times \hat{d}}$. With a mask $M \in \{0, -\infty\}^{n \times n}$, the dense attention output at position i can be written as follows.

$$y_i = \sum_{j=0}^{n-1} a_{ij} v_j, \quad \text{where } a_{ij} = \frac{\exp\left(q_i^\top k_j / \sqrt{\hat{d}} + M_{ij}\right)}{\sum_{t=0}^{n-1} \exp\left(q_i^\top k_t / \sqrt{\hat{d}} + M_{it}\right)}, \text{ for } i, j = 0, \dots, n-1. \quad (3.1)$$

For bidirectional attention, all nonpadding positions are allowed. For causal attention, $M_{ij} = -\infty$ whenever $j > i$. The cost of computing the score matrix, normalizing it, and multiplying by V is

$$T_{\text{dense}} = \mathcal{O}(n^2 \hat{d}), \quad S_{\text{dense}} = \mathcal{O}(n^2), \quad (3.2)$$

where T_{dense} denotes time complexity and S_{dense} denotes space, or memory, complexity. These estimates are up to constants from heads, batches, and implementation details. This is the same bottleneck discussed in Section 2.2.1, but it is useful to restate it here because the rest of the chapter changes the structure of this computation.

The quadratic term becomes limiting in the regimes that motivate this thesis project. In language modeling, n is the context length. Increasing context from 1024 to 4096 multiplies the dense attention matrix size by a factor of sixteen. This affects training memory, because intermediate activations are needed for backpropagation, and it affects inference, because every new query must interact with a growing key value cache. In vision, the input is often a two dimensional grid. If an image feature map has height H and width W and is flattened into $n = HW$ tokens, dense attention scales as

$$\mathcal{O}(n^2 \hat{d}) = \mathcal{O}(H^2 W^2 \hat{d}). \quad (3.3)$$

Doubling both spatial dimensions multiplies the number of patches by four and the attention matrix by sixteen. This is a direct obstacle for dense prediction tasks such as segmentation, where preserving spatial resolution is important.

One response is to remove most attention edges. Local windows, strided patterns, random edges, global tokens, clustering, and pooling all reduce the number of pairwise comparisons. These methods are useful, and they define much of the literature reviewed in Section 2.2. The main tradeoff is that they change how information moves through the model. A strict local window has linear cost, but a token at one end of the input cannot affect a token at the other end in a single layer. Shifted windows and strided patterns reduce this path length, but global communication still depends on the chosen graph. Pooling keeps a global summary, but it may discard fine grained information before a query has a chance to select what it needs.

The target in this chapter is therefore not only lower asymptotic cost. The target is a particular kind of lower cost. Local information should remain available at full resolution, because nearby tokens or patches often contain the most detailed evidence. Long distance information should also remain available in every layer, because language and vision both require global context. In text, a later token may depend on an earlier entity, topic, or syntactic choice. In images, a local patch may be ambiguous without object level or scene

level information. This suggests an operator that uses different resolutions for different distances rather than using one uniform resolution for all pairs.

This is the point where the connection to fast summation methods becomes useful. In an n body problem, a direct method evaluates the effect of every source particle on every target particle. The Fast Multipole Method avoids this uniform treatment by separating interactions into near field and far field components [42, 102]. Nearby sources are treated directly. Distant sources are grouped and represented by compact expansions that can be reused by many targets. The analogy is not exact. Attention does not begin with a known analytic kernel, and its weights are learned through a downstream task rather than chosen to satisfy a prescribed approximation error. The useful idea is the decomposition. It is wasteful to represent every distant source at the same resolution as a neighbouring source if distant interactions are compressible.

Fast Multipole Attention applies this idea to the attention matrix rather than to a physical potential. For each query position i , the allowed key positions are conceptually split into a near field and several far field groups,

$$\{j : M_{ij} = 0\} = \mathcal{N}(i) \cup \bigcup_{\ell=1}^L \mathcal{F}_{\ell}(i), \quad (3.4)$$

where $\mathcal{N}(i)$ denotes nearby positions and $\mathcal{F}_{\ell}(i)$ denotes distant positions represented at level ℓ of a hierarchy. The near field will be evaluated with ordinary attention. The far field will be represented by learned summaries of groups of keys and values. Section 3.2 gives the precise construction. At this stage, Equation (3.4) should be read as a design principle. FMA replaces one dense all pairs matrix by a structured set of local blocks and multilevel summarized blocks.

The geometry of the input matters for this split. A one dimensional sequence has an intrinsic order. Token i has natural neighbours near index i , and longer range context lies farther away along the sequence. A two dimensional image feature map has a spatial grid. A patch has neighbours above, below, left, and right, and larger regions of the image can be summarized at coarser spatial scales. Flattening an image into a one dimensional list hides this geometry from the attention pattern. FMA uses a one dimensional hierarchy for sequences and a two dimensional hierarchy for images. The attention weights are still learned from content. The hierarchy only determines which interactions are computed at fine resolution and which interactions are routed through summaries.

The desired attention operator should satisfy several design goals.

1. **Keep direct local interactions.** Nearby tokens or patches should attend to each other without being compressed first. This preserves the detailed comparisons that

are important for local syntax, local texture, boundaries, and small objects. It also keeps the method close to standard attention in the region where attention scores are often most concentrated.

2. **Preserve a global receptive field.** Every output position should have access to information from the whole allowed input in a single layer. The access may be through summaries rather than individual keys and values, but there should not be disconnected regions whose information can only arrive after many layers. This separates FMA from purely local windows.
3. **Represent distant context at multiple resolutions.** A single global pooled vector is too coarse for many tasks. It mixes all distant positions before the query can choose among them. A hierarchy gives each query several scales of far field information. Coarser levels cover larger regions, while finer levels retain more location detail.
4. **Learn the summaries end to end.** Classical fast multipole methods use analytic expansions of known kernels. A Transformer layer does not have such a fixed kernel. Its useful summaries depend on the data, the layer, and the task. The aggregation maps in FMA should therefore be trainable, so the model can learn what information from a distant group should be exposed to queries.
5. **Reduce both computation and memory.**
 The method should avoid materializing the dense $n \times n$ attention matrix. Loglinear FMA uses fine queries at every far field level. Linear FMA further reduces interactions by downsampling queries. Section 3.3 counts interactions, summary construction, and output combination.
6. **Support standard Transformer masks.** The same attention mechanism should work in bidirectional encoder settings and causal decoder settings. This requirement is important because the language experiments in this chapter include both masked language modeling and autoregressive modeling. The near field and far field definitions must respect the mask, so that a causal model never uses information from future positions.
7. **Extend naturally from sequences to images.** The one dimensional and two dimensional operators should follow the same principle. The implementation details differ because a sequence uses intervals and an image uses spatial cells, but the mathematical structure should remain the same. Local interactions are exact, distant

interactions use learned multilevel summaries, and the total number of hierarchy cells is linear in the number of input tokens.

These goals also clarify what FMA is not trying to do. It is not an exact algorithm for dense attention. Exact IO aware methods such as FlashAttention [30, 32] keep the mathematical operator unchanged and improve how it is executed on hardware. FMA changes the operator by imposing a multilevel structure on the attention pattern. It is also not a task specific window design. The same near field plus far field principle is used for language and vision. Finally, it is not a direct transfer of FMM error analysis to neural attention. The connection to the Fast Multipole Method is an organizing principle for compression and reuse. The effectiveness of the learned operator must be tested empirically.

The rest of the chapter follows this problem statement. Section 3.2 defines the one dimensional and two dimensional FMA operators. Section 3.3 analyzes the resulting computation and memory costs. The experimental sections then test whether this design gives a useful accuracy and efficiency tradeoff on language modeling, image classification, semantic segmentation, and object detection.

3.2 Fast Multipole Attention Method

This section defines one dimensional and two dimensional FMA through a common operator. Both variants fix the maximum input size before training. Both downsample keys and values across the hierarchy. Loglinear FMA retains fine queries at every level. Linear FMA also downsamples queries. Section 3.3 compares their complete costs.

3.2.1 Common Notation

The input to one attention head is $X \in \mathbb{R}^{n \times d}$. Linear maps produce

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \quad (3.5)$$

where $Q, K, V \in \mathbb{R}^{n \times \hat{d}}$. The row vectors are denoted q_i, k_i, v_i . The index set is written as I . For a sequence, $I = \{0, \dots, n-1\}$. For an image feature map, $I = \{0, \dots, H-1\} \times \{0, \dots, W-1\}$ and $n = HW$. The notation $j \in I$ therefore means either a token index or a spatial grid point, depending on the setting.

FMA uses a tree partition of I . The leaves of the tree are base cells of size r . In one dimension these cells are intervals of length r . In two dimensions they are squares of side

length r . Coarser cells are obtained by doubling side length at each level. Let $\mathcal{C}_\ell$ be the collection of cells at level ℓ , where $\ell = 1$ is the base level. A cell $C \in \mathcal{C}_\ell$ contains $\mathcal{O}(r2^{\ell-1})$ points in one dimension and $\mathcal{O}(r^2 4^{\ell-1})$ points in two dimensions, apart from boundary cells.

Each far field cell is replaced by learned key and value representatives. The number of representatives per cell is the far field rank p . For $C \in \mathcal{C}_\ell$ and $s = 0, \dots, p-1$, define

$$\bar{k}_{C,s}^{(\ell)} = \sum_{j \in C} \pi_{\ell,s}^K(j; C) k_j, \quad \bar{v}_{C,s}^{(\ell)} = \sum_{j \in C} \pi_{\ell,s}^V(j; C) v_j. \quad (3.6)$$

The coefficients $\pi_{\ell,s}^K$ and $\pi_{\ell,s}^V$ are trainable. They are shared across cells at the same level, with the same relative position inside the cell using the same coefficient. For $p = 1$, each cell has one aggregated key and one aggregated value. Larger p gives a richer low rank description of the cell. The base construction in the paper is written with $p = 1$; the rank- p extension used in the main experiments is included here directly.

It is useful to stack all summaries at level ℓ into matrices

$$\bar{K}_\ell = R_\ell^K K, \quad \bar{V}_\ell = R_\ell^V V, \quad (3.7)$$

where R_ℓ^K and R_ℓ^V are sparse learned aggregation matrices. Their rows correspond to pairs (C, s) with $C \in \mathcal{C}_\ell$ and $s = 0, \dots, p-1$. The matrices in Equation (3.7) are a compact way to express the same operation as Equation (3.6). They also make the connection to hierarchical matrix structure explicit.

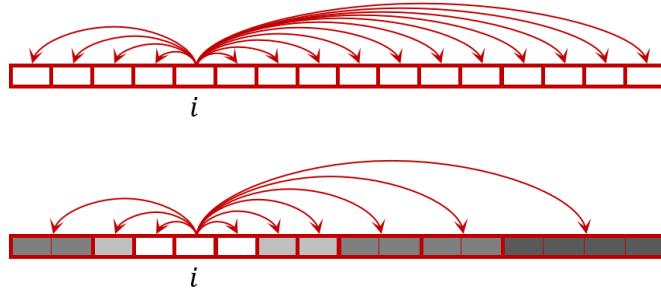


Figure 3.1: Conceptual view of one dimensional FMA. Full attention computes a separate interaction between query token i and every source token. FMA keeps exact attention in a three-cell near field of width $3r$ and represents the far field by learned summaries of progressively larger groups.

Figure 3.1 illustrates the interaction pattern for one query. In full attention, the query interacts separately with all n source tokens. FMA retains this fine resolution within the

near field. In the far field, consecutive tokens are aggregated into groups of increasing size, such as r , $2r$, and $4r$. Each far field group is represented by learned key and value summaries, so the query computes a small number of interactions per hierarchy level rather than one interaction per far field token. Near field detail is therefore preserved while far field context remains available through coarser representations.

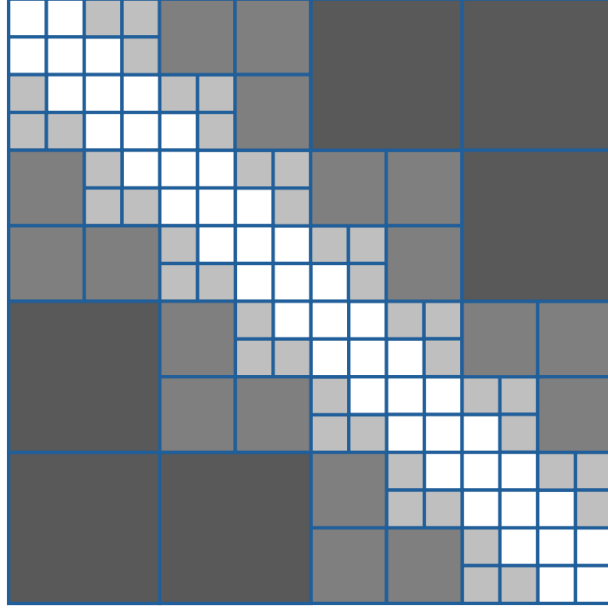


Figure 3.2: Sparsified attention matrix for one dimensional FMA. White cells denote exact near field interactions. Gray blocks denote summarized far field interactions, with darker shades indicating coarser hierarchy levels.

Figure 3.2 shows the corresponding structure across all queries. The white diagonal band contains the exact near field. Farther from the diagonal, the far field is partitioned into increasingly large blocks at progressively coarser resolutions. The pattern covers the full matrix, so every query retains a global receptive field, but the far field is stored and evaluated through learned low rank summaries. This block hierarchy is closely related to the data sparse structure of hierarchical matrices, which compress off diagonal blocks while retaining detailed interactions near the diagonal [10, 48].

3.2.2 Near Field and Far Field Decomposition

For each query position i , FMA chooses a near field $\mathcal{N}(i) \subset I$ and far field interaction lists $\mathcal{I}_\ell(i) \subseteq \mathcal{C}_\ell$ for $\ell = 1, \dots, L-1$. The near field contains nearby input positions and is computed at full resolution. The far field lists contain cells whose key and value summaries are used instead of their individual positions. The construction is chosen so that

$$\{j \in I : M_{ij} = 0\} = \mathcal{N}(i) \cup \bigcup_{\ell=1}^{L-1} \bigcup_{C \in \mathcal{I}_\ell(i)} C. \quad (3.8)$$

The sets on the right hand side are pairwise disjoint. The mask M is the task mask. In a bidirectional layer it removes only padding. In a causal layer it removes future positions. Equation (3.8) says that every allowed source position is represented exactly once, either as an individual near field token or as part of one far field cell.

This cover gives a simple way to read the method. FMA does not pool the whole sequence into one vector. For each query, every source position is used at one resolution. Nearby positions stay as individual tokens. Farther positions are placed into cells. The same source token may be near field for one query and far field for another.

The near field contribution is standard masked attention restricted to $\mathcal{N}(i)$,

$$\alpha_{ij} = \frac{\exp(q_i^\top k_j / \sqrt{\hat{d}})}{\sum_{t \in \mathcal{N}(i)} \exp(q_i^\top k_t / \sqrt{\hat{d}})}, \quad j \in \mathcal{N}(i). \quad (3.9)$$

For a far field level ℓ , query i attends to the representatives $\bar{k}_{C,s}^{(\ell)}$ of the cells in $\mathcal{I}_\ell(i)$,

$$\beta_{i,C,s}^{(\ell)} = \frac{\exp(q_i^\top \bar{k}_{C,s}^{(\ell)} / \sqrt{\hat{d}})}{\sum_{D \in \mathcal{I}_\ell(i)} \sum_{t=0}^{p-1} \exp(q_i^\top \bar{k}_{D,t}^{(\ell)} / \sqrt{\hat{d}})}, \quad (3.10)$$

where $C \in \mathcal{I}_\ell(i)$ and $s = 0, \dots, p-1$. The FMA output is then

$$y_i = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} v_j + \sum_{\ell=1}^{L-1} \sum_{C \in \mathcal{I}_\ell(i)} \sum_{s=0}^{p-1} \beta_{i,C,s}^{(\ell)} \bar{v}_{C,s}^{(\ell)}. \quad (3.11)$$

This equation is the main definition of the attention operator. It says that nearby positions contribute through exact values, while distant cells contribute through learned summaries.

It is helpful to split Equation (3.11) into scale contributions,

$$y_i^{(0)} = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} v_j, \quad y_i^{(\ell)} = \sum_{C \in \mathcal{I}_\ell(i)} \sum_{s=0}^{p-1} \beta_{i,C,s}^{(\ell)} \bar{v}_{C,s}^{(\ell)}. \quad (3.12)$$

Then

$$y_i = y_i^{(0)} + \sum_{\ell=1}^{L-1} y_i^{(\ell)}. \quad (3.13)$$

The term $y_i^{(0)}$ is the local update. Each $y_i^{(\ell)}$ is the contribution from one farther scale. The layer adds these terms before the usual output projection and residual connection. Thus the query receives local tokens and several far field summaries, not just one global pooled vector.

The same definition can be written in matrix form. Let M_0 be the mask that keeps only near field entries. For $\ell \geq 1$, let M_ℓ be the mask that keeps only columns corresponding to representatives of cells in $\mathcal{I}_\ell(i)$ for each row i . Define

$$A_0 = \text{softmax}_{\text{row}} \left(\frac{QK^\top}{\sqrt{\hat{d}}} + M_0 \right), \quad (3.14)$$

$$A_\ell = \text{softmax}_{\text{row}} \left(\frac{Q\bar{K}_\ell^\top}{\sqrt{\hat{d}}} + M_\ell \right), \quad \ell = 1, \dots, L-1. \quad (3.15)$$

The output matrix is

$$Y_{\text{FMA}} = A_0 V + \sum_{\ell=1}^{L-1} A_\ell \bar{V}_\ell. \quad (3.16)$$

Equivalently,

$$Y_{\text{FMA}} = \left(A_0 + \sum_{\ell=1}^{L-1} A_\ell R_\ell^V \right) V. \quad (3.17)$$

The matrix in parentheses is the attention matrix acting on values, but it is not formed in memory. The product $A_\ell R_\ell^V$ represents far field blocks with rank at most p per cell. This is the matrix view of the exact near field and the low rank far field.

The softmax in Equations (3.14) and (3.15) is applied separately at each resolution level. FMA is therefore not just sparse dense attention. Each level produces its own contribution. If all near and far representatives were placed in one softmax, coarse summaries

and local tokens would compete in the same normalization. The current choice keeps each level separate. The resulting matrix in Equation (3.17) is not a row stochastic dense attention matrix. It is a learned feature mixing operator built from local blocks and far field summaries. The following sections give the interaction lists for sequences and images.

3.2.3 One Dimensional Construction

For a sequence, take $I = \{0, \dots, n-1\}$. The dyadic hierarchy for length n uses

$$h_\ell = r2^{\ell-1}, \quad L = \lceil \log_2(n/r) \rceil. \quad (3.18)$$

Each variant sets a maximum length $n_{\max}$ before training and fixes $L = L_0 = \lceil \log_2(n_{\max}/r) \rceil$ for $n \leq n_{\max}$. Cell sizes and learned aggregation parameters stay fixed. Cells are truncated at the input boundary. For $\ell = 1, \dots, L-1$, the level ℓ cells are intervals

$$C_{\ell,m} = \{j \in I : mh_\ell \leq j < (m+1)h_\ell\}, \quad m = 0, \dots, M_\ell - 1, \quad (3.19)$$

where $M_\ell = \lceil n/h_\ell \rceil$. For position i , define the base-cell index and the coarser block indices by

$$b_0(i) = \lfloor i/r \rfloor, \quad b_\ell(i) = \lfloor i/(r2^\ell) \rfloor, \quad \ell \geq 1. \quad (3.20)$$

The near field is the three base cell window around i ,

$$\mathcal{N}_{\text{1D}}(i) = \{j \in I : |b_0(j) - b_0(i)| \leq 1\}. \quad (3.21)$$

Thus $|\mathcal{N}_{\text{1D}}(i)| \leq 3r$, apart from boundary effects and task masks.

Tokens enter the far field at the finest level at which their coarse block lies within the three-block neighbourhood but they have not already appeared at a finer resolution,

$$\mathcal{F}_\ell^{\text{1D}}(i) = \left\{ j \in I : |b_\ell(j) - b_\ell(i)| \leq 1, j \notin \mathcal{N}_{\text{1D}}(i) \cup \bigcup_{1 \leq m < \ell} \mathcal{F}_m^{\text{1D}}(i) \right\}, \quad \ell = 1, \dots, L-1. \quad (3.22)$$

The interaction list at level ℓ is the set of level ℓ cells contained in this shell,

$$\mathcal{I}_\ell^{\text{1D}}(i) = \{C_{\ell,m} \in \mathcal{C}_\ell : C_{\ell,m} \subseteq \mathcal{F}_\ell^{\text{1D}}(i)\}. \quad (3.23)$$

The near field and the far field sets form the disjoint cover used in Equation (3.8), after the usual boundary truncation.

For each level $\ell = 1, \dots, L-1$ and representative $s = 0, \dots, p-1$, FMA learns key and value coefficient vectors $\boldsymbol{\pi}_{\ell,s}^K, \boldsymbol{\pi}_{\ell,s}^V \in \mathbb{R}^{h_\ell}$. The entries $\pi_{\ell,s,u}^K$ and $\pi_{\ell,s,u}^V$ are trainable, where $u = 0, \dots, h_\ell - 1$ denotes the relative position within a cell. For a cell $C_{\ell,m} \in \mathcal{C}_\ell$, the one dimensional summaries are

$$\bar{k}_{C_{\ell,m},s}^{(\ell)} = \sum_{j \in C_{\ell,m}} \pi_{\ell,s,j-mh_\ell}^K k_j, \quad \bar{v}_{C_{\ell,m},s}^{(\ell)} = \sum_{j \in C_{\ell,m}} \pi_{\ell,s,j-mh_\ell}^V v_j. \quad (3.24)$$

Here $j - mh_\ell$ is the relative position of token j within the cell. The same coefficient vectors are therefore reused for every cell at level ℓ . The row formula in Equation (3.11) with $\mathcal{N}(i) = \mathcal{N}_{\text{1D}}(i)$ and $\mathcal{I}_\ell(i) = \mathcal{I}_\ell^{\text{1D}}(i)$ defines one dimensional FMA.

This construction gives each query full resolution access to nearby tokens and a constant number of summarized cells at every coarser level. The number of summary representatives per query is $\mathcal{O}(pL)$, while the exact near field has size at most $3r$. The complexity consequences are derived in Section 3.3.

3.2.4 Two Dimensional Construction

For images, take $I = \{0, \dots, H-1\} \times \{0, \dots, W-1\}$. The formulas are simplest when $H = W$, but the construction extends to rectangular feature maps by using the same cells in each coordinate and truncating at the boundary. The full quadtree uses

$$h_\ell = r2^{\ell-1}, \quad L = \lceil \log_2(\max\{H, W\}/r) \rceil. \quad (3.25)$$

Both variants replace $\max\{H, W\}$ by the maximum spatial extent set before training. Each resulting depth L_0 stays fixed within its supported range. For $\ell = 1, \dots, L-1$, the level ℓ cells are rectangles

$$C_{\ell,u,v} = \{(x, y) \in I : uh_\ell \leq x < (u+1)h_\ell, vh_\ell \leq y < (v+1)h_\ell\}. \quad (3.26)$$

For a pixel or patch $i = (x_i, y_i)$, define

$$b_\ell^x(i) = \lfloor x_i / (r2^\ell) \rfloor, \quad b_\ell^y(i) = \lfloor y_i / (r2^\ell) \rfloor, \quad \ell \geq 0. \quad (3.27)$$

The near field is the 3×3 window of base cells around i ,

$$\mathcal{N}_{\text{2D}}(i) = \{j \in I : |b_0^x(j) - b_0^x(i)| \leq 1, |b_0^y(j) - b_0^y(i)| \leq 1\}. \quad (3.28)$$

It contains at most $9r^2$ positions. The base cell is $r \times r$. The near field is $3r \times 3r$.

As in one dimension, a position is assigned at the first coarse level at which it becomes available outside the exact near field. The far field and interaction list are

$$\mathcal{F}_\ell^{2D}(i) = \left\{ j \in I : |b_\ell^x(j) - b_\ell^x(i)| \leq 1, |b_\ell^y(j) - b_\ell^y(i)| \leq 1, \right. \\ \left. j \notin \mathcal{N}_{2D}(i) \cup \bigcup_{1 \leq m < \ell} \mathcal{F}_m^{2D}(i) \right\}, \quad (3.29)$$

$$\mathcal{I}_\ell^{2D}(i) = \{C_{\ell,u,v} \in \mathcal{C}_\ell : C_{\ell,u,v} \subseteq \mathcal{F}_\ell^{2D}(i)\}, \quad \ell = 1, \dots, L-1. \quad (3.30)$$

Thus each level contributes a ring of cells around the query, measured in the ℓ_∞ grid distance on the quadtree. Near cells are handled exactly, and far cells are represented by summaries.

The two dimensional summaries can be learned as full two dimensional kernels. In the implementation used for the experiments, they are separable in the two coordinates. For $C_{\ell,u,v} \in \mathcal{C}_\ell$ and $s = 0, \dots, p-1$,

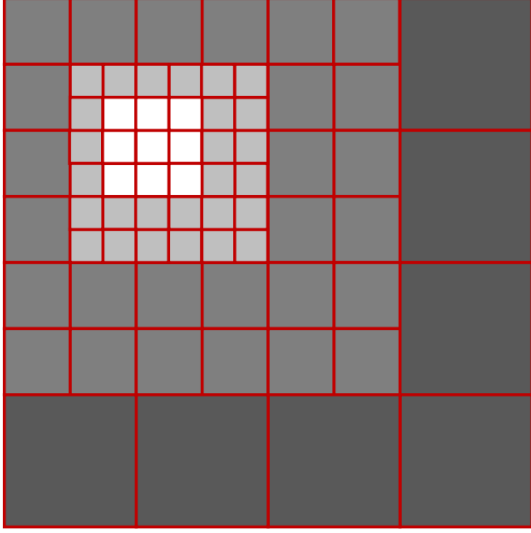
$$\pi_{\ell,s}^K(a, b) = \pi_{\ell,s}^{K,x}(a) \pi_{\ell,s}^{K,y}(b), \\ \pi_{\ell,s}^V(a, b) = \pi_{\ell,s}^{V,x}(a) \pi_{\ell,s}^{V,y}(b), \quad (3.31)$$

for offsets $0 \leq a, b < h_\ell$. The summaries are

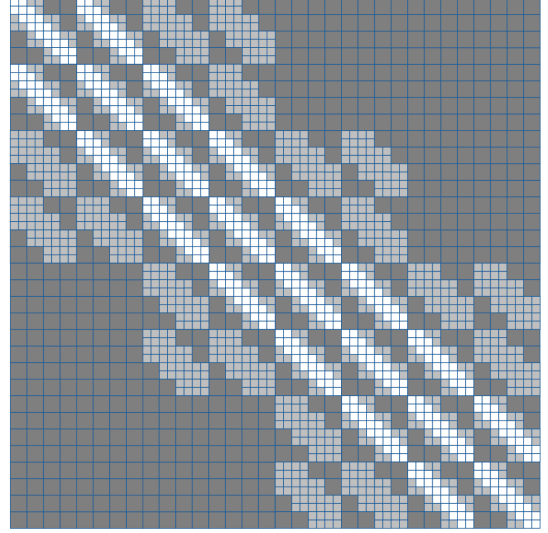
$$\bar{k}_{C_{\ell,u,v},s}^{(\ell)} = \sum_{(x,y) \in C_{\ell,u,v}} \pi_{\ell,s}^K(x - uh_\ell, y - vh_\ell) k_{(x,y)}, \\ \bar{v}_{C_{\ell,u,v},s}^{(\ell)} = \sum_{(x,y) \in C_{\ell,u,v}} \pi_{\ell,s}^V(x - uh_\ell, y - vh_\ell) v_{(x,y)}. \quad (3.32)$$

Using separable kernels reduces the number of aggregation parameters while preserving the spatial structure of the cell. The row formula in Equation (3.11) with $\mathcal{N}(i) = \mathcal{N}_{2D}(i)$ and $\mathcal{I}_\ell(i) = \mathcal{I}_\ell^{2D}(i)$ defines two dimensional FMA.

We visualize the quadtree partition and the corresponding sparsified attention matrix in Figure 3.3.



(a) Two dimensional quadtree with four levels.



(b) Two dimensional attention pattern with three levels.

Figure 3.3: FMA two dimensional hierarchy and a corresponding sparsified attention matrix. White cells denote exact near field interactions. Gray cells denote summarized far field interactions, with darker shades indicating coarser hierarchy levels. Panel (b) corresponds to the top left quarter of panel (a).

The important difference from applying one dimensional FMA to a flattened image is the geometry of the partition. A flattened sequence imposes an arbitrary row major order. A quadtree keeps horizontal and vertical neighbourhoods together. Therefore the exact near field is isotropic on the image grid, and the far field summaries correspond to square spatial regions rather than intervals in a flattened list.

3.2.5 Causal Constraints and Far Field Capacity

The formulas above are written for the bidirectional case. Padding is handled by removing padded positions from the near field and from all far field cells. The causal case uses the same operator with the additional constraint $j \leq i$ in one dimensional sequence order. Equivalently, every level mask M_ℓ removes any representative whose source cell contains future positions. Boundary regions that cannot be represented by an allowed coarse cell are handled at a finer level. This preserves the disjoint cover of allowed positions and prevents future information from entering the output.

The causal case is the only place where the hierarchy needs extra care. A coarse cell can straddle the boundary between past and future positions. Such a cell cannot be used as one representative, because its summary would mix allowed and forbidden tokens. For query i , define the allowed prefix

$$P(i) = \{j \in I : j \leq i\}. \quad (3.33)$$

A far field cell C is valid for query i only if

$$C \subseteq P(i). \quad (3.34)$$

If a cell fails this condition, its allowed part is represented by smaller children until the cover reaches cells that are entirely in the past, or by individual near field tokens at the boundary. Thus the causal mask is part of the hierarchy construction. It decides which coarse cells may appear in the interaction list. This lets the same FMA operator be used in bidirectional encoders and autoregressive decoders.

The rank parameter p controls the capacity of the far field representation. When $p = 1$, each far field cell contributes one aggregated key and one aggregated value. When $p > 1$, the cell contributes p representatives, and the query can assign different attention weights to different learned modes of the same cell. The value side of a far field block then has rank at most p before the output projection. Increasing p increases the number of score evaluations and the memory for summaries by a proportional factor. The ablation study later in the chapter tests this tradeoff.

The rank interpretation can be made explicit. Fix a far field cell C at level ℓ and collect its p value representatives as rows of

$$\bar{V}_C^{(\ell)} = \begin{bmatrix} \bar{v}_{C,1}^{(\ell)} \\ \vdots \\ \bar{v}_{C,p}^{(\ell)} \end{bmatrix} \in \mathbb{R}^{p \times \hat{d}}. \quad (3.35)$$

For all queries that use this cell at level ℓ , the contribution of C has the form

$$Y_C^{(\ell)} = B_C^{(\ell)} \bar{V}_C^{(\ell)}, \quad (3.36)$$

where the rows of $B_C^{(\ell)}$ contain the attention weights $\beta_{i,C,s}^{(\ell)}$. Therefore

$$\text{rank}(Y_C^{(\ell)}) \leq p. \quad (3.37)$$

This has the same form as a low rank far field block in a hierarchical matrix. The learned aggregation kernels give the vectors in $\bar{V}_C^{(\ell)}$. The queries give the coefficients in $B_C^{(\ell)}$.

The level by level normalization in Equation (3.16) is a design choice. One could put all near and far representatives into one list and apply one softmax. This version instead normalizes each resolution separately. It keeps the near field and each far field scale as separate terms in the output. It also matches the implementation used in the experiments.

This normalization also affects training. The near field usually has more representatives than any one far field level, especially in two dimensions where the exact region can contain up to $9r^2$ patches. With one softmax, a large near field could dominate the smaller far field lists. With levelwise normalization, each scale chooses among its own candidates. The following linear projection then mixes the scale outputs. LayerNorm and the residual path control the size of the final update.

FMA is therefore a learned hierarchical attention layer. It is not claiming that a dense attention matrix has a fixed analytic multipole expansion. The assumption is simpler. Nearby tokens should be kept exactly, and distant tokens can often be represented by learned summaries. The next section gives the cost of this layer and explains the loglinear and linear variants.

3.3 Complexity and Implementation

Complexity We count work per attention head. The base cell size r , rank $p \geq 1$, and head width $\hat{d}$ are fixed as n grows. The batch size, number of heads, number of Transformer blocks, embedding width, and feedforward width are also fixed. Both variants fix the maximum input size and hierarchy depth $L = L_0$ before training. The formulas retain L to compare different maximum sizes. The processed length n includes any padding evaluated by the kernels.

For the one dimensional construction, the near field contains at most $3r$ tokens. At each coarser level, the shell around the query contains only a constant number of intervals. Each interval contributes p representatives. Thus the number of score vectors seen by one query is bounded by

$$m_{1D}(i) \leq 3r + c_{1D}pL, \quad (3.38)$$

where c_{1D} depends on the interaction list. For loglinear FMA, interaction work and score storage satisfy

$$T_{1D} = \mathcal{O}\left(n\hat{d}(r + pL)\right), \quad S_{1D} = \mathcal{O}(n(r + pL)). \quad (3.39)$$

Across models with increasing $n = n_{\max}$, $L = \lceil \log_2(n/r) \rceil$ gives $\mathcal{O}(n \log n)$. With fixed $L = L_0$, these bounds are linear within the supported range.

In Equation (3.39), nr counts exact local interactions and npL counts far field interactions. Larger r widens the exact region. Larger p increases the summary capacity.

In two dimensions, the near field contains at most $9r^2$ positions. Each far field level contributes a bounded number of cells. With $n = HW$,

$$m_{2D}(i) \leq 9r^2 + c_{2D}pL. \quad (3.40)$$

For loglinear FMA, interaction and score storage costs are

$$T_{2D} = \mathcal{O}\left(n\hat{d}(r^2 + pL)\right), \quad S_{2D} = \mathcal{O}\left(n(r^2 + pL)\right). \quad (3.41)$$

For inputs at growing maximum resolutions, these bounds are loglinear in n . Fixed $L = L_0$ gives linear bounds within the supported range.

The two dimensional near field term grows with r^2 rather than r . This is expected because local geometry is now area rather than length. A larger base cell gives a wider exact spatial neighbourhood, which can help boundaries and small objects. It also increases the exact local attention cost quickly. Thus the two dimensional version follows the same idea as the one dimensional version, but r affects memory more strongly.

Direct evaluation of the summaries in Equations (3.24) and (3.32) costs $\mathcal{O}(np\hat{d})$ per level in both variants. The cells partition the input at each level. The total is $T_{KV} = \mathcal{O}(npL\hat{d})$, including separable two dimensional aggregation.

Fine queries produce n vectors per level. Their direct sum in Equation (3.13) costs $T_{\text{combine}} = \mathcal{O}(nL\hat{d})$. Direct lifting and summation of coarse outputs have the same bound for a bounded number of coarse vectors per fine position. Replication and interpolation with fixed support satisfy this condition.

Both variants summarize keys and values. Linear FMA also downsamples queries. Each query uses $\mathcal{O}(p)$ far field representatives per level. For fixed $r, p, \hat{d}$, compare models at dyadic lengths $n = n_{\max}$ with $L = \lceil \log_2(n/r) \rceil$. The one dimensional counts are

Linear FMA	Loglinear FMA	
Downsampled queries	Full resolution queries	
$n, n/2, n/4, \dots$	$n, n, n, \dots$	
$n \sum_{\ell=0}^{L-1} 2^{-\ell} < 2n$	$\sum_{\ell=0}^{L-1} n = nL$	
$T_{\text{far}} = \mathcal{O}(np\hat{d}) = \mathcal{O}(n)$	$T_{\text{far}} = \mathcal{O}(npL\hat{d}) = \mathcal{O}(n \log n)$	(3.42)

On square two dimensional grids, query counts decrease by four per level. The sum $n \sum_{\ell=0}^{L-1} 4^{-\ell} < 4n/3$ remains linear. These counts exclude summary construction and output assembly.

Both variants retain the near field cost $\mathcal{O}(nrd\hat{d})$. Let $T_{\text{int},v}$ count near and far field interactions for variant v . The direct total cost is

$$T_{\text{total},v} = T_{\text{int},v} + \mathcal{O}(npL\hat{d}) + \mathcal{O}(nL\hat{d}).$$

For linear FMA, the summary term also includes query downsampling. In two dimensions, replace r by r^2 .

Both variants support sequences with $n \leq n_{\text{max}} \leq r2^{L_0}$. Each sequence spans at most two adjacent blocks of width $r2^{L_0-1}$ at the coarsest neighbourhood scale. The near field and far field shells therefore cover all positions (Equation (3.8)). Shorter inputs use boundary masks. Empty far field lists contribute zero. Causal attention covers the allowed prefix (Section 3.2.5). In two dimensions, the condition is $\max\{H, W\} \leq r2^{L_0}$.

For fixed $r, p, \hat{d}, L_0$, both variants have linear total cost for $n \leq n_{\text{max}}$. The depth L_0 grows logarithmically with n_{max} . At $n = n_{\text{max}}$, loglinear and linear FMA have interaction costs $\mathcal{O}(n \log n)$ and $\mathcal{O}(n)$, respectively. The direct total cost is $\mathcal{O}(n \log(n_{\text{max}}/r))$ for either variant, or $\mathcal{O}(n \log n)$ at $n = n_{\text{max}}$.

The effective matrix in Equation (3.17) is never materialized. In one dimension, fine queries store $\mathcal{O}(n(r + pL))$ scores and coarse queries store $\mathcal{O}(n(r + p))$. Replace r by r^2 in two dimensions. Stored summaries use $\mathcal{O}(p\hat{d} \sum_{\ell} |\mathcal{C}_{\ell}|)$ storage. A running output sum needs an $\mathcal{O}(n\hat{d})$ buffer. Retaining all lifted outputs uses $\mathcal{O}(nL\hat{d})$ storage. Backpropagation may retain further tensors. With fixed depth, these costs are linear within the supported range.

The attention term is only one part of a Transformer block. The query, key, and value projections, the output projection, and the feedforward network still have costs that are linear in sequence length and quadratic in hidden width. At short context lengths these dense channel operations can dominate runtime. FMA is most useful when the attention matrix itself becomes the bottleneck, which happens for long language contexts and high resolution feature maps. This is why the experimental results report both accuracy and memory across several sequence lengths and image resolutions.

The hierarchy summaries add parameters, but their number is small compared with the dense projection matrices in a typical Transformer. In one dimension, a level with cell size h_{ℓ} stores aggregation weights over relative positions inside a cell. In two dimensions, separable aggregation kernels keep the number of aggregation parameters proportional to

side length rather than area. The main cost of FMA is therefore not the parameter count. It is the runtime and activation storage for the local blocks, far field scores, and summaries.

An FMA layer has four main steps. First, the input is projected to Q, K, V as in standard multihead attention. Second, learned aggregation maps build key and value summaries for all cells and levels. Third, the layer computes exact attention in the near field and summarized attention in the far field. Fourth, the near and far contributions are added and passed through the output projection. The far field blocks are built from summaries only when needed. Forming the full blocks explicitly would remove the memory saving.

Implementation Our TVM schedule and CUDA kernels [23] compute near field attention, summaries, far field interactions, and output combination. They do not materialize dense off diagonal blocks. FMA-linear additionally downsamples queries. The cost analysis includes all these stages.

Figure 3.4 summarizes the implementation of one dimensional FMA. Queries and keys are aggregated across hierarchy levels. The near field scores are stored in the sparse banded matrix A , while $B^{(\ell)}$ denotes the far field interactions at level ℓ . The value representatives are combined with these scores to produce level specific outputs, which are accumulated into the final output X . Darker blocks denote coarser hierarchy levels.

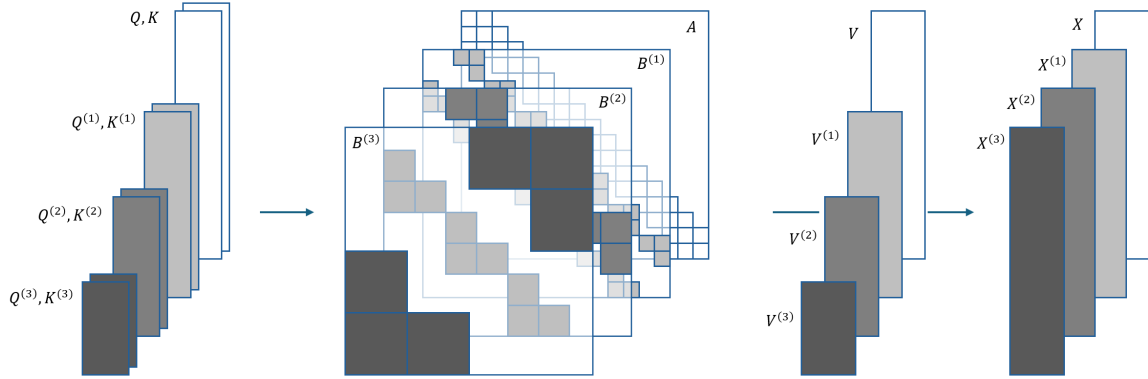


Figure 3.4: Overview of the one dimensional FMA implementation. Queries, keys, and values are aggregated hierarchically, and near field and far field interactions are evaluated without materializing dense low rank off diagonal blocks.

Figure 3.5 gives a more detailed view of the same computation. The first stage constructs multilevel representations of Q , K , and V . The second stage computes the exact

near field blocks and the summarized far field blocks at their respective resolutions. Their conceptual superposition gives the hierarchical attention pattern, although the full matrix is not formed in memory. Specialized CUDA kernels apply these blocks directly to the value representations, delivering the theoretical memory and arithmetic benefits of FMA in the GPU implementation.

The hierarchy gives the implementation fewer pairwise interactions, but this does not guarantee lower wall clock time at every input length. The current FMA kernels are less mature than common exact attention kernels. Exact IO aware methods such as FlashAttention accelerate dense attention without changing its quadratic arithmetic count. These ideas are complementary: the near field of FMA is standard attention on short blocks, and the far field reductions also have a regular tiled form suitable for further kernel optimization.

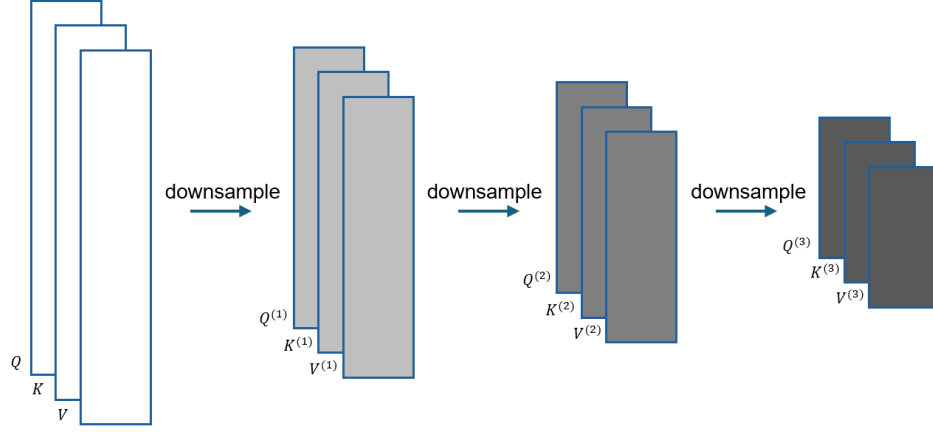
3.4 Relationship to Prior Attention Methods

Chapter 2 reviewed efficient attention broadly. This section only states how FMA differs from the closest mechanisms.

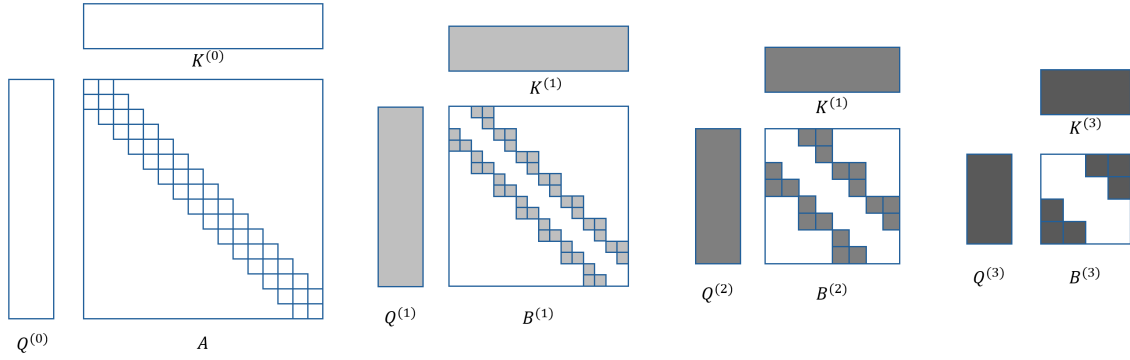
Sparse attention methods reduce cost by selecting a subset of token pairs. Local windows, dilated patterns, and global tokens give explicit connectivity graphs [11, 186]. These patterns are useful, but the choice of edges determines how information can move. A purely local window needs several layers before distant positions can interact. Adding global tokens gives a short path, but the global information is compressed through a small fixed set of positions. FMA makes a different choice. It keeps exact local attention and adds global access through many spatially organized summaries. Every layer has a global receptive field, but distant information is represented at coarser resolution.

Linearized attention uses feature maps for associative evaluation [27, 66]. GLA adds input dependent gates to recurrent matrix memory [179]. DeltaNet revises stored associations with delta updates, and Gated DeltaNet combines these updates with gating [180, 181]. Mamba uses selective state space dynamics [44]. Mamba-2 connects a selective state space layer with structured masked attention [31]. These models support parallel training and recurrent decoding with a state size independent of sequence length. FMA keeps softmax attention in the near field and over explicit summaries at each far field level. These five alternatives are relevant comparators, especially for autoregressive modeling. Their performance relative to FMA is not tested here.

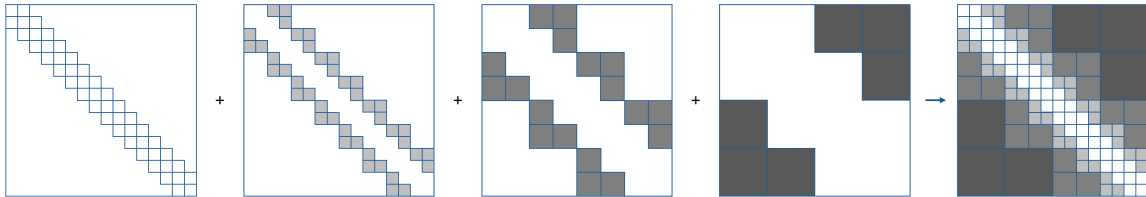
Pooling and memory compressed attention reduce the key and value length before



(a) Multilevel aggregation of queries, keys, and values.



(b) Near field matrix A and far field matrices $B^{(\ell)}$ at successively coarser levels.



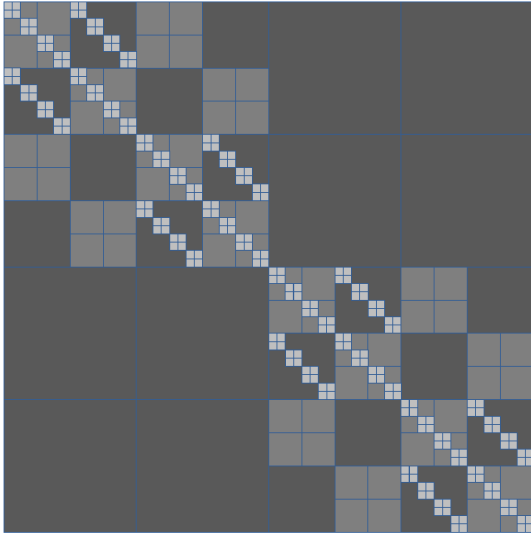
(c) Conceptual superposition of the near field and far field blocks.

Figure 3.5: Construction of the bidirectional FMA operator with four hierarchy levels. The multilevel representations in panel (a) produce the sparse score blocks in panel (b). Panel (c) shows their conceptual superposition as the final hierarchical attention pattern; the implementation applies these blocks without materializing the full matrix.

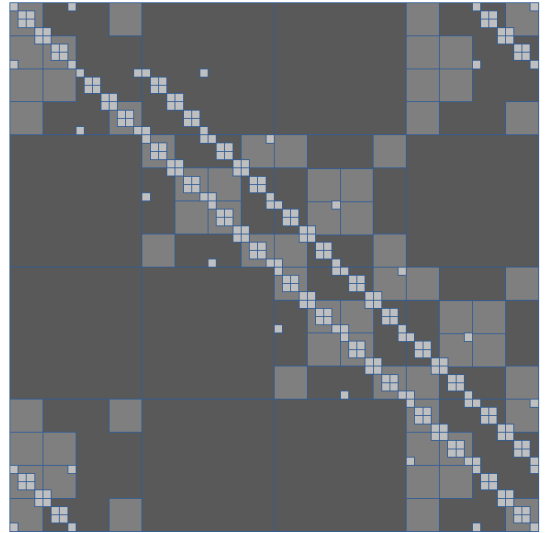
attention [96, 174]. This can be efficient, but it often applies the same compression to all queries at a scale. A query then cannot choose among fine positions inside a compressed region. FMA uses pooling only for distant regions. Nearby regions remain uncompressed, and far field summaries are learned separately at multiple levels. The ablation in Section 3.8 shows that learned aggregation performs better than uniform average pooling in the tested language model.

Hierarchical attention methods are closest in spirit. H-Transformer-1D uses a hierarchical matrix inspired approximation with low rank off diagonal blocks [192]. MRA attention recursively decomposes interactions using multiresolution structure [187]. FMA shares the idea that far field interactions can be compressed, but it uses trainable aggregation kernels for keys and values and provides both one dimensional and two dimensional constructions. The two dimensional version uses a quadtree rather than a flattened sequence, so the geometry of image patches is preserved.

Windowed vision Transformers such as Swin compute attention inside local image windows and shift the window partition between layers [99]. Figure 3.6 overlays the resulting attention patterns across three Swin stages.



(a) Unshifted local windows.



(b) Shifted local windows.

Figure 3.6: Attention sparsity across three stages of a Swin Transformer. Different shades denote different stages; when cells overlap, the lightest shade, corresponding to the finest resolution, is shown. Unshifted layers attend within fixed local windows, while shifted layers allow cross window exchange along the displaced window boundaries.

Panel (a) shows the block diagonal patterns produced by the first unshifted layer of each stage. As patch resolution decreases across stages, the union of these local patterns forms progressively wider diagonal bands. Panel (b) adds the shifted layers, allowing information to cross the original window boundaries. Nevertheless, every individual Swin layer remains local, so global communication must accumulate through multiple layers and patch merging stages. FMA2D uses a different mechanism. Each FMA2D layer retains an exact local near field and simultaneously accesses the rest of the image through multilevel quadtree summaries, giving a global receptive field in one layer. Its square near field and quadtree far field also yield a more isotropic connectivity pattern than the stitched diagonal structure in panel (b). The comparison is therefore not only about cost, but also about whether global context is available within a single attention layer or emerges through a sequence of local layers.

Finally, exact IO aware attention such as FlashAttention is orthogonal to FMA [30, 32]. FlashAttention computes the same dense attention result with better memory traffic. FMA computes a different hierarchical attention operator with fewer interactions. When dense attention fits on hardware and exactness is required, an IO aware exact kernel is the right tool. When length or resolution makes the dense operator too costly, FMA offers a structured approximation with local exactness and global summarized context.

Table 3.1 provides a structured comparison of FMA with the most closely related efficient attention mechanisms, highlighting the distinguishing features of each approach.

Method	Near field exact	Far field aggregation	Global per layer	Causal support	Input geometry	Complexity
Swin [99]	✓	Shifting	–	–	2D	$\mathcal{O}(n)$
P2T [174]	–	Average pool	✓	–	2D	$\mathcal{O}(n)$
LRFormer [188]	–	Projection	✓	–	2D	$\mathcal{O}(n)$
HAT-Net [98]	✓	Average pool	✓	–	2D	$\mathcal{O}(n)$
CSWin [37]	✓	Cross stripe	Partial	–	2D	$\mathcal{O}(n\sqrt{n})$
H-Transformer [192]	✓	Fixed average	✓	–	1D	$\mathcal{O}(n)$
MRA [187]	Adaptive	Adaptive	✓	–	1D	$\mathcal{O}(n \log n)$
FMA	✓	Learned	✓	✓	1D and 2D	$\mathcal{O}(n \log n)/\mathcal{O}(n)$

Table 3.1: Feature comparison of efficient attention mechanisms most relevant to FMA. Near field exact indicates whether exact attention is computed for nearby tokens or pixels. Far field aggregation describes how distant context is represented. Global per layer indicates whether each attention layer has a global receptive field. A dash indicates that the feature is not supported.

The FMA entries in Table 3.1 describe interaction work and score storage as $n = n_{\max}$ grows, with $r, p, \hat{d}$ fixed. Section 3.3 also counts summary construction and output assembly.

The comparison highlights the combination that distinguishes FMA. It preserves exact near field interactions while learning multilevel far field summaries, rather than relying on fixed averaging, a predefined projection, or window shifting. It provides global context within every layer and supports both bidirectional and causal attention. The one dimensional interval hierarchy and two dimensional quadtree use the same construction, and the underlying partition and aggregation principle extends in a direct way to higher dimensional inputs. Thus FMA combines learned multilevel summaries, a global receptive field, and loglinear or linear complexity in both sequence and image settings, while its geometric hierarchy can in principle be generalized to any dimension.

3.5 Experimental Setup

The experiments test FMA in both one dimensional and two dimensional settings. The one dimensional tasks are autoregressive character modeling on enwik8 [101] and masked language modeling on WikiText-103 [106]. The two dimensional tasks are ImageNet-1K image classification [34] and ADE20K semantic segmentation [189], followed by COCO 2017 object detection [94]. A separate symbolic retrieval task tests access to specific distant values (Section 3.9).

The language experiments use a common six layer Transformer in fairseq [112], with width 768 and 12 heads. The baselines cover dense attention (Full), key and value compression (Memory Compressed Attention [96]), sparse connectivity (Reformer [69] and BigBird [186]), linearization (Linear Transformer [66]), and hierarchical approximation (H-Transformer-1D [192] and MRA [187]). This selection compares complementary attention designs within the same backbone. Training hyperparameters are tuned on Full and held fixed for the alternatives. The tables specify the baselines used for each task. For FMA, the default far field rank is $p = 4$. The ablation later varies both the base cell size r and rank p .

The autoregressive model is trained with a causal mask. It predicts the next character and is evaluated by bits per character (bpc). Lower bpc means better negative log likelihood. The masked language model uses a bidirectional encoder and randomly masks 15% of tokens. It is evaluated by perplexity. Lower perplexity means better word prediction.

The image classification experiments replace standard attention blocks in ViT backbones with two dimensional FMA blocks. The base models follow ViT-B/16 and ViT-L/16 [38]: the base model has width 768, MLP width 3072, and 12 layers, while the large model has width 1024, MLP width 4096, and 24 layers. The main comparisons are against

ViT and Swin [99], with published P2T and CSWin values included as context in the table where available [37, 174]. Models are trained for 300 epochs with a cosine learning rate schedule peaking at 10^{-3} . The base cell side length is $r = 2$ at image size 224 and $r = 4$ at image size 384. The reported metric is single crop top 1 validation accuracy.

For semantic segmentation, FMA2D replaces the self attention blocks in SegFormer-B4 and SegFormer-B5 style backbones [175]. The models are pretrained on ImageNet-22K and finetuned on ADE20K for 160,000 iterations with 512×512 crops using AdamW, weight decay 0.01, an initial learning rate of 6×10^{-5} , and polynomial decay. Performance is measured by mean intersection over union (mIoU). Higher mIoU indicates better dense prediction.

For object detection, FMA2D replaces every attention block in a ViT-B/16 backbone used by Faster R-CNN [129]. A simple feature pyramid and detection head follow the plain-ViT design of ViTDet [93]. The paper reports both a run initialized from a full-attention ImageNet-1K checkpoint and a run trained from scratch. This distinction is important because the pretrained weights were not learned with the FMA operator.

Memory numbers report peak training GPU allocation for the corresponding experiment. They include forward activations and backward storage. Some quadratic language baselines require gradient accumulation to fit long contexts. In those cases, the table records the number of passes and the peak memory per pass. Comparisons should be read within each table, because the hardware and training configuration differ across task families. “Full” denotes dense softmax attention under the task mask. The reported setup does not identify its exact backend or confirm FlashAttention use. FlashAttention avoids storing the full score matrix and changes memory traffic [30, 32]. The measurements therefore describe the tested implementations and do not establish an advantage over FlashAttention.

3.6 Language Modeling Results

3.6.1 Autoregressive Language Modeling

Table 3.2 reports character level language modeling results on enwik8 at context lengths from 512 to 4096. Lower bits per character (bpc) is better. The comparison includes quadratic full and memory compressed attention, together with subquadratic Reformer, Linear Transformer, BigBird, H-Transformer, and the two FMA variants. Memory is peak

Model	Context	Test bpc ↓	Memory (GB)
Full	512	1.346	16.5
MCA	512	1.439	13.5
Reformer	512	1.372	15.5
Linear	512	1.424	38.2
BigBird	512	1.465	16.4
H-Transformer	512	1.853	12.3
FMA	512	1.353	<i>11.6</i>
FMA-linear	512	<i>1.391</i>	9.8
Full	1024	1.236	46.5
MCA	1024	1.351	27.4
Reformer	1024	<i>1.271</i>	31.8
Linear	1024	1.343	74.9
BigBird	1024	1.345	23.1
H-Transformer	1024	1.789	23.7
FMA	1024	1.256	<i>21.6</i>
FMA-linear	1024	1.298	17.6
Full	2048	1.128	2×71.5
MCA	2048	1.276	73.4
Reformer	2048	1.252	62.9
Linear	2048	1.274	2×72.1
BigBird	2048	1.281	39.5
H-Transformer	2048	1.684	<i>38.4</i>
FMA	2048	1.220	48.2
FMA-linear	2048	<i>1.244</i>	34.1
Full	4096	1.091	8×61.7
MCA	4096	1.212	4×49.2
Reformer	4096	1.180	2×63.6
Linear	4096	1.207	4×71.0
BigBird	4096	1.216	72.2
H-Transformer	4096	1.535	66.0
FMA	4096	1.133	2×50.5
FMA-linear	4096	<i>1.154</i>	<i>69.4</i>

Table 3.2: Autoregressive enwik8 results. Bold values are best among the subquadratic methods for a context length, and italics indicate second best. A multiplier in the memory column denotes the number of gradient accumulation passes.

GPU consumption. A multiplier indicates that gradient accumulation is required to match the target batch size because the model does not fit in one pass.

The loglinear FMA variant gives the lowest bpc among all efficient methods at every context length. Relative to the next best efficient model, Reformer, its bpc improvement is approximately 1.4%, 1.2%, 2.6%, and 4.1% for contexts 512, 1024, 2048, and 4096, respectively. The advantage therefore becomes more pronounced as the context grows. At the same time, loglinear FMA remains among the four lowest memory efficient methods, while FMA-linear is best or second best at the shorter context lengths.

Full attention remains the most accurate method. The measured Full implementation needs two gradient accumulation passes at 2048 tokens and eight at 4096. Several other baselines also require multiple passes at the longest context. FMA remains close to full attention while using two passes at 4096, and FMA-linear fits the same context in one pass. The linear variant gives up a small amount of accuracy relative to loglinear FMA in exchange for lower memory at 512–2048 tokens and removal of gradient accumulation at 4096. Compared with H-Transformer, FMA-linear achieves substantially lower bpc with comparable memory. FMA-linear reduces coarse interactions within the configured maximum context. Section 3.3 gives the cost of extending this range.

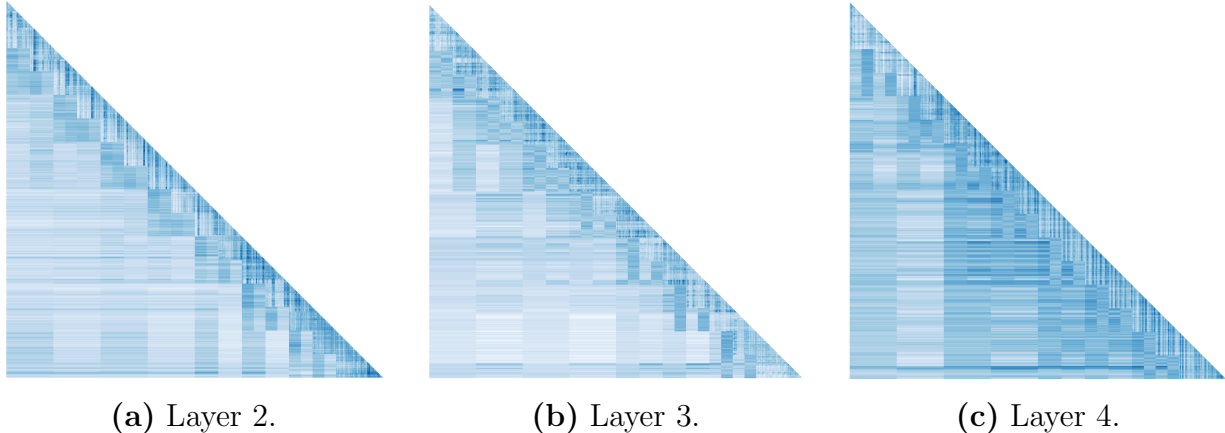


Figure 3.7: Sample FMA attention maps from different layers and heads of the autoregressive enwik8 model with context length 1024. The empty upper triangles result from the causal mask.

Figure 3.7 confirms the intended near field and far field decomposition after training. High magnitude coefficients are concentrated near the diagonal, where FMA retains token level resolution. Away from the diagonal, progressively larger blocks reflect the coarser

groups used for the far field. The locations and intensities of these blocks vary across layers and heads, showing that the model learns to use both detailed local interactions and selected long range dependencies rather than reducing the far field to uniform pooling.

3.6.2 Bidirectional Language Modeling

Table 3.3 gives the complete masked language modeling comparison on WikiText-103. FMA again has the best perplexity among the efficient variants at each context length. At context length 2048, it reaches perplexity 8.95, close to the full-attention value 8.70, while avoiding the two-pass quadratic memory footprint. FMA-linear reduces memory further and remains competitive with the other efficient baselines.

Model	Context	Test perplexity ↓	Memory (GB)
Full	512	9.21	15.4
Reformer	512	11.03	14.6
Linear	512	14.95	11.8
MRA	512	11.82	11.1
H-Transformer	512	10.60	10.7
FMA	512	9.45	10.9
FMA-linear	512	10.78	8.2
Full	1024	8.89	36.8
Reformer	1024	10.56	27.4
Linear	1024	14.24	22.0
MRA	1024	10.20	17.5
H-Transformer	1024	9.81	16.9
FMA	1024	9.06	18.6
FMA-linear	1024	10.13	15.6
Full	2048	8.70	2×56.5
Reformer	2048	9.90	49.2
Linear	2048	14.06	48.1
MRA	2048	9.66	35.3
H-Transformer	2048	9.45	33.1
FMA	2048	8.95	37.0
FMA-linear	2048	9.64	29.5

Table 3.3: Masked language modeling results on WikiText-103. Bold values are best among the subquadratic methods for a context length.

These language results support two conclusions. First, the learned far field summaries keep enough information for these tasks. The model stays close to dense attention while

storing fewer interactions. Second, the linear variant is useful when memory is the main constraint, but the loglinear variant gives a better accuracy and memory balance when both variants fit.

The autoregressive and bidirectional tasks test different masks. In the autoregressive setting, the hierarchy must respect the causal prefix for every query. Far field summaries near the causal boundary cannot mix future tokens, so some coarse cells are replaced by finer cells. FMA remains competitive at long context lengths under this constraint. In the bidirectional setting, the hierarchy can use both left and right context. The WikiText-103 results show that the same far field summaries work in both settings.

Full attention remains the reference operator when it is computationally feasible. FMA changes the operator, so the goal is not to reproduce every dense attention coefficient. The goal is to keep enough local and distant information for the downstream loss while reducing the number of interactions. The language tables show this tradeoff most clearly at longer context lengths.

3.7 Vision Results

The vision experiments test whether the two dimensional quadtree construction is useful beyond one dimensional sequences. The main issue is spatial resolution. A ViT style global attention block has direct access to all patches but incurs quadratic attention cost. A Swin style block is efficient but limits attention to local windows inside each layer. FMA2D keeps exact local interactions and gives each layer global access through quadtree summaries.

3.7.1 Image Classification

Table 3.4 reports ImageNet-1K classification results without ImageNet-22K pretraining. At 224^2 , FMA2D-B reaches 84.0% top 1 accuracy with lower peak training memory than ViT-B/16 and Swin-B. At 384^2 , FMA2D-B remains competitive with CSWin-B published accuracy and uses less measured memory than ViT-B/16 and Swin-B in the matched runs. At the large model scale, FMA2D-L reaches 87.5% top 1 accuracy at 384^2 with lower measured memory than ViT-L/16 and Swin-L.

Table 3.5 shows the corresponding ImageNet-22K pretrain setting. FMA2D-B improves to 87.1% at 384^2 , and FMA2D-L reaches 88.8%. The same memory pattern remains. The advantage is most relevant at high resolution, where global dense attention uses a large fraction of training memory.

Model	Params	Image size	Top 1	Memory
ViT-B/16 [38]	86M	224	81.8	17.2 GB
Swin-B [99]	88M	224	83.5	15.3 GB
P2T-L [174]	55M	224	83.9	n/a
CSWin-B [37]	78M	224	84.2	n/a
FMA2D-B	92M	224	84.0	14.6 GB
ViT-B/16	86M	384	83.6	41.4 GB
Swin-B	88M	384	84.5	28.1 GB
CSWin-B [37]	78M	384	85.4	n/a
FMA2D-B	85M	384	85.3	26.6 GB
ViT-L/16	307M	384	85.2	103.5 GB
Swin-L	197M	384	87.0	66.8 GB
FMA2D-L	201M	384	87.5	63.4 GB

Table 3.4: ImageNet-1K single crop classification without ImageNet-22K pretraining. Top 1 is measured in percent. Published P2T and CSWin values are included for context, and their memory is not directly comparable.

Model	Params	Image size	Top 1	Memory
ViT-B/16	86M	384	84.0	41.4 GB
Swin-B	88M	384	86.4	28.1 GB
CSWin-B [37]	78M	384	87.0	n/a
FMA2D-B	85M	384	87.1	26.6 GB
ViT-L/16	307M	384	85.8	103.5 GB
Swin-L	197M	384	88.1	66.8 GB
CSWin-L [37]	173M	384	87.5	n/a
FMA2D-L	201M	384	88.8	63.4 GB

Table 3.5: ImageNet-1K single crop classification with ImageNet-22K pretraining. Top 1 is measured in percent. Published CSWin values are included for context.

FMA2D avoids dense off diagonal blocks and uses less memory in these comparisons. The memory values in Tables 3.4 and 3.5 were measured with distributed data parallel training on four A100 GPUs and an effective batch size of 128. Fixed depth gives linear storage within the supported spatial range (Section 3.3).

We also examine the model predictions qualitatively. Figure 3.8 shows predictions from FMA2D-B on ImageNet-1K validation images at 224×224 . The model correctly classifies a diverse range of categories with high confidence, typically above 96%. The errors involve visually similar categories, such as magpie versus bald eagle and bulletproof vest versus military uniform. These examples indicate that the hierarchical attention mechanism preserves fine grained visual features.

FMA2D-B ImageNet-1K Sample Predictions (224x224)

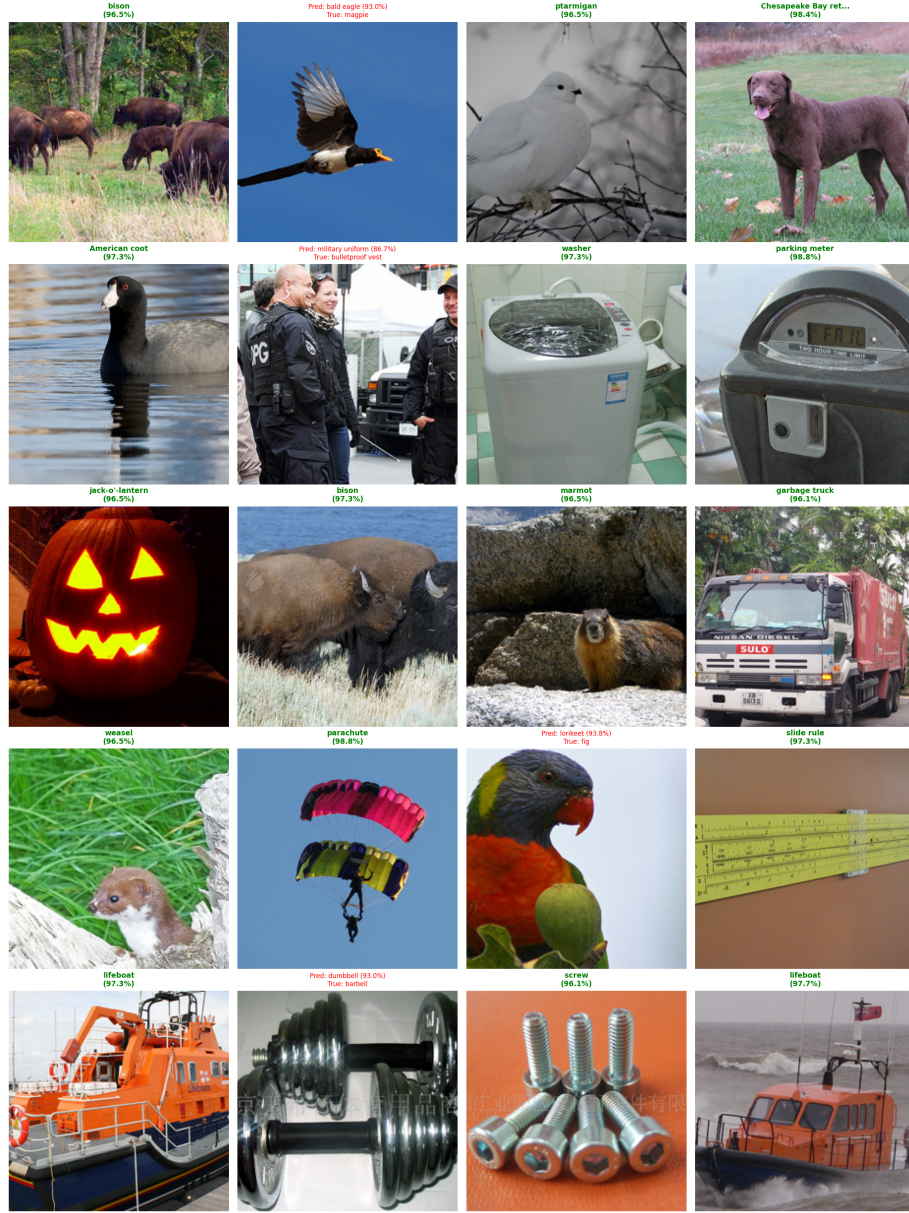


Figure 3.8: Sample predictions from FMA2D-B on the ImageNet-1K validation set at 224×224 . Green labels indicate correct predictions with confidence scores. Red labels show misclassifications with the predicted and true classes.

3.7.2 Semantic Segmentation

The segmentation results in Table 3.6 test dense prediction, where local boundary detail and global context are both important. Replacing SegFormer attention blocks with FMA2D improves mIoU at both tested scales. FMA2D-L-SegFormer reaches 52.4 mIoU, compared with 51.5 for SegFormer-B5. It also exceeds the listed Swin-L-UPerNet result while using far fewer parameters. The result suggests that the quadtree summaries do not only help classification. They also support pixel level prediction.

Segmentation tests the two dimensional operator in a different way from classification. Classification can rely on a global representation after many layers. Segmentation needs local spatial detail until the final dense prediction head. FMA2D fits this requirement. The local near field keeps neighbouring patches at full resolution, while quadtree summaries provide scene level context. This helps explain why replacing attention inside a segmentation backbone can improve mIoU rather than only reduce memory.

The comparison with windowed attention is mainly about receptive field structure. Swin obtains global communication by stacking shifted local windows and merging patches across stages. FMA2D gives every block access to global context through summaries. The experiments do not show that one design is always better. They show that FMA2D is competitive and can help when high resolution memory is a limiting factor.

Method	Params	mIoU
Swin-B-UPerNet [99]	121M	48.1
Swin-L-UPerNet [99]	234M	50.5
SegFormer-B4 [175]	64M	51.2
SegFormer-B5 [175]	85M	51.5
FMA2D-B-SegFormer	71M	51.9
FMA2D-L-SegFormer	90M	52.4

Table 3.6: ADE20K validation results with ImageNet-22K pretraining. mIoU is measured in percent.

3.7.3 Object Detection

The COCO 2017 experiment tests FMA2D in a third vision setting. Both models use a Faster R-CNN detector with a ViT-B/16 backbone and a ViTDet-style feature pyramid. FMA2D replaces self attention in all 12 backbone blocks. The pretrained run is finetuned

for 36 epochs from an ImageNet-1K checkpoint trained with full attention; the from-scratch run uses a 12-epoch schedule. Table 3.7 reports validation average precision.

Initialization	Attention	AP $\uparrow$	AP ₅₀ $\uparrow$	AP ₇₅ $\uparrow$
ImageNet-1K	Full	39.7	61.0	42.8
ImageNet-1K	FMA2D	38.2	59.1	40.7
From scratch	Full	9.7	21.4	7.6
From scratch	FMA2D	11.6	23.9	9.9

Table 3.7: COCO 2017 validation results for Faster R-CNN. The ImageNet-1K initialization was trained using full attention before the attention operator was replaced.

With full-attention pretraining, full attention leads FMA2D by 1.5 AP. This comparison is not a matched pretraining test: replacing the operator introduces a mismatch between the pretrained backbone weights and the new hierarchical attention blocks. When both backbones are trained from scratch, FMA2D instead improves AP from 9.7 to 11.6. The absolute from-scratch values are low, but the comparison isolates the attention mechanism and suggests that the spatial hierarchy can provide a useful inductive bias at detection resolution.

The pretrained experiment is analogous to a drop in replacement test. The backbone weights were learned with full attention during ImageNet pretraining, so replacing the attention operator with FMA2D introduces a mismatch that the 36 fine tuning epochs only partially overcome. This explains the 1.5 AP gap in the pretrained setting. A backbone pretrained with FMA2D would provide weights already adapted to the hierarchical decomposition and would likely reduce this gap.

Figure 3.9 shows the validation AP trajectories. When training from scratch, FMA2D leads full attention and the difference grows over the course of training, indicating that the final improvement is not caused by a single noisy evaluation. With the full attention pretrained backbone, full attention retains its advantage throughout fine tuning, consistent with the operator mismatch described above.

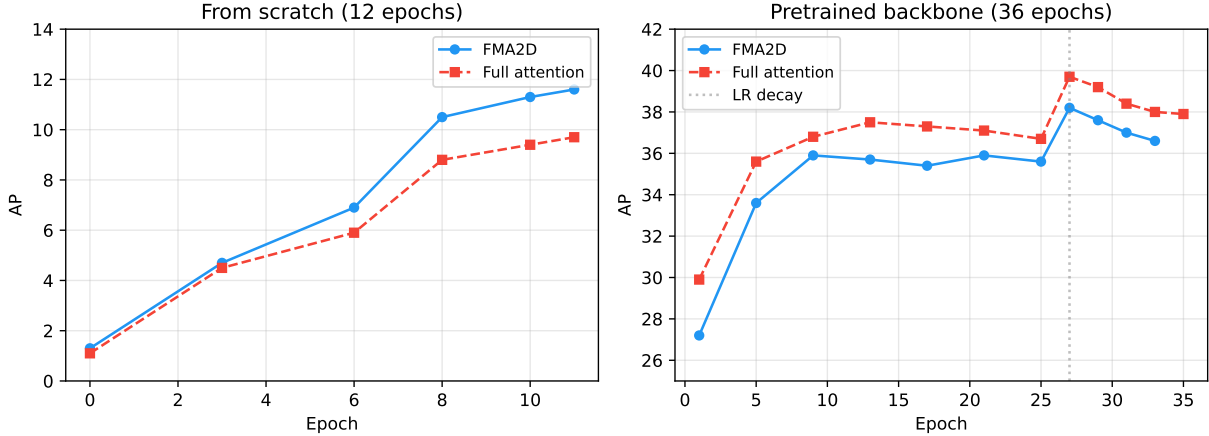


Figure 3.9: COCO 2017 validation AP during training. Left: when trained from scratch, FMA2D leads full attention and widens the gap. Right: with a pretrained full attention backbone, full attention retains the advantage.

3.8 Ablations and Approximation Analysis

Ablation Study The main architecture choices in FMA are the base cell size r and the far field rank p . Larger r increases the exact near field. Larger p gives each far field cell more learned representatives. Both changes increase memory and computation. Table 3.8 reports an enwik8 ablation at context length 1024 with the same six layer language model used above.

Rank	$r = 16$	$r = 32$	$r = 64$
Avg pool	1.354, 18.6 GB	1.346, 18.6 GB	1.341, 18.6 GB
$p = 1$	1.321, 18.1 GB	1.279, 19.1 GB	1.270, 20.5 GB
$p = 2$	1.320, 18.4 GB	1.272, 19.3 GB	1.258, 20.9 GB
$p = 4$	1.307, 19.1 GB	1.266, 19.7 GB	1.256, 21.6 GB
$p = 8$	1.276, 20.2 GB	1.262, 21.1 GB	1.251, 23.2 GB

Table 3.8: Ablation on enwik8 with context length 1024. Each entry reports bpc followed by peak training memory. Avg pool replaces learned far field summaries by uniform average pooling.

The ablation has three clear trends. First, increasing p usually lowers bpc because the far field has more expressive capacity. Second, increasing r also improves bpc because

more context is handled exactly near the query. Third, the memory cost grows with both choices, as predicted by Equation (3.39).

The ablation also shows that r and p play different roles. Increasing r widens the exact local band. It does not make a distant cell more expressive. Increasing p gives each far field cell more modes. It does not add more exact local comparisons. A model that needs sharp local detail may benefit more from larger r . A model that needs richer distant context may benefit more from larger p . This is why both parameters are kept in the construction.

Table 3.8 also includes an average pool baseline, which replaces the learned far field basis functions with uniform average pooling. At context length 1024 and $r = 16, 32, 64$, average pooling gives a higher bpc than every learned variant. Even a single learned basis function with $p = 1$ outperforms uniform averaging, and the accuracy generally improves further as the rank increases with only a modest additional memory cost. This comparison demonstrates the advantage of learned aggregation over static pooling and supports the use of trainable far field bases in FMA.

Efficiency Analysis Figure 3.10 benchmarks isolated attention layers from 256 to 8192 tokens with batch size 64 and embedding dimension 768. The measured Full and compressed implementations show quadratic memory and latency growth and exceed 40 GB by a sequence length of 4096. Over the measured range, FMA-linear tracks the baselines with linear complexity and remains competitive in inference latency at 8192 tokens. Section 3.3 separates the interaction savings from the total cost. The benchmark isolates attention. Full model throughput also includes projections, MLPs, normalization, and embeddings.

End-to-end throughput gives another view of the tradeoff. In the ImageNet-1K 384×384 setting, FMA2D-B uses less training memory than ViT-B/16 but has lower wall clock throughput in the current implementation. Table 3.9 shows this point. At batch size 8, FMA2D-B reaches about 71% of ViT-B/16 training throughput and about 74% of inference throughput, while using less peak training memory and giving higher top 1 accuracy. The throughput gap is an implementation issue as well as an algorithmic issue. The FMA kernels are less optimized than standard dense attention kernels.

Empirical Approximation Power Analysis The approximation analysis asks a different question. It tests whether the FMA hierarchy has enough capacity to mimic a trained dense attention layer. The experiment uses a six-layer bidirectional WikiText-103 model with width 768, 12 heads, context length 1024, $r = 64$, and $p = 4$, trained for 10,000 steps.

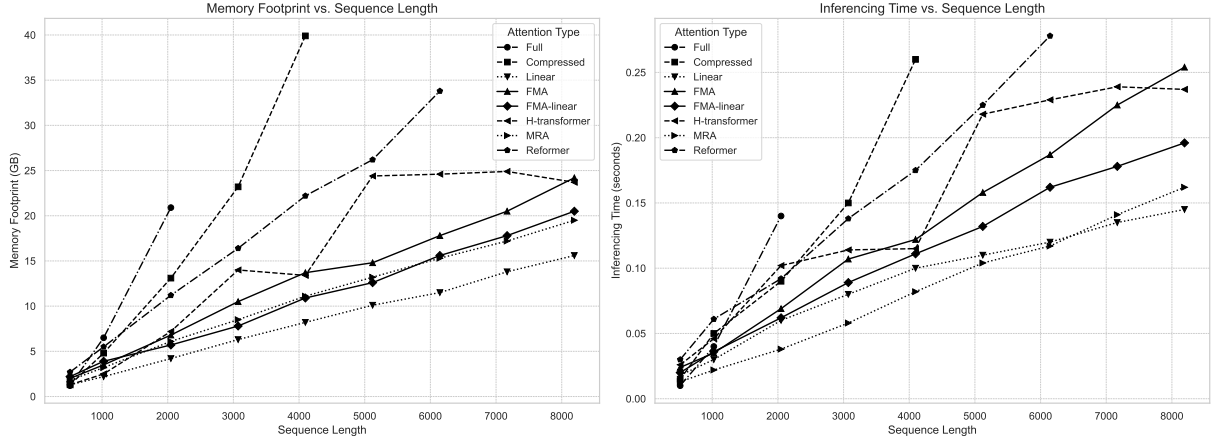


Figure 3.10: Efficiency comparison of attention variants across input lengths. Left: peak GPU memory footprint. Right: inference latency.

Model	Batch	Train img/s	Infer img/s	Train mem	Infer mem	Top 1
ViT-B/16	8	255.4	763.3	5.4 GB	2.0 GB	84.5
ViT-B/16	4	145.2	663.6	3.3 GB	1.8 GB	84.5
FMA2D-B	8	181.6	561.9	3.8 GB	1.9 GB	85.3
FMA2D-B	4	111.9	514.0	2.6 GB	1.8 GB	85.3

Table 3.9: End to end throughput on ImageNet-1K at 384×384 .

A full-attention Transformer is first trained. Then, using its trained Q, K, V projections on validation data, only the FMA aggregation kernels are optimized to minimize the output difference between FMA and dense attention. Table 3.10 reports the best relative L_2 error and cosine similarity by layer.

Layer	Relative L_2	Cosine similarity
0	0.154	0.987
1	0.180	0.982
2	0.294	0.968
3	0.275	0.973
4	0.408	0.939
5	0.335	0.952

Table 3.10: Best case FMA1D approximation of trained dense attention on WikiText-103.

The cosine similarity remains between 0.939 and 0.987 across layers. This does not prove that FMA is an accurate approximation for every possible dense attention map. It does show that, for the trained model and data tested here, the learned hierarchy can approximate the dense attention outputs closely.

The layer dependence in Table 3.10 is expected. Early layers often contain local lexical and positional information, which the exact near field can preserve well. Later layers combine more abstract features, and their attention patterns may depend less on geometric distance. The relative error is therefore larger in some deeper layers. The cosine similarity remains high, so the main output direction is still close to the dense attention output.

A second test replaces dense attention in the trained model by FMA and then finetunes only the FMA aggregation kernels. The full attention model has validation perplexity 8.87. The replacement model reaches perplexity 9.12 after tuning only the FMA kernels. This is a small fraction of the total parameters. The result supports the claim that the hierarchy is a useful inductive bias, not merely a way to reduce memory.

This replacement experiment is stricter than training FMA from scratch. Most model weights are fixed at values learned for dense attention, and only the aggregation kernels can change. Recovering most of the perplexity under this constraint shows that the FMA basis can match much of the dense attention output. Training the whole model with FMA then lets the query, key, value, and feedforward weights adapt to the hierarchical operator from the beginning.

3.9 Symbol Retrieval

We test whether FMA can retrieve a specific value from distant positions in a needle in a haystack task. Both causal Transformers are trained from scratch. This is not a natural language evaluation or a test of the language model checkpoints above.

Task. Each input contains random distractors, **TAG**, a target v , more distractors, and a final **QUERY**. Distractors and v are independent uniform draws from $\{0, \dots, 255\}$. The special symbols are **TAG** = 256 and **QUERY** = 257. For total length n and depth $d \in [0, 100]$, the marker has zero based position $s = \text{round}((d/100)(n - 3))$. The target and query occupy $s + 1$ and $n - 1$. The model predicts only v at the query. No answer token or teacher forcing is used. Chance accuracy is $1/256 = 0.390625\%$.

Models. Both models have three Pre-LN layers, width 128, four heads, FFN width 512 with GELU, sinusoidal positions, and no dropout. A final LayerNorm and linear head classify the query state into 256 symbols. Input embeddings and output weights are untied. Q/K/V/O projections and the classifier have no bias. FFN projections use bias. Common weights are initialized identically within each of seeds 0, 1, 2. FMA adds aggregation parameters, so total parameter counts differ.

Full uses PyTorch causal `scaled_dot_product_attention`. This does not identify the backend used in the earlier language benchmarks. FMA uses learned K/V summaries with $r = 32, p = 4$, query chunks of 256, and generic PyTorch kernels without Triton. The exact near field covers the current base block up to the query and the preceding block, not a fixed 32 token window. The supported maximum is fixed at 8192. This retrieval implementation activates $L(n) = \max(1, \lceil \log_2(n/r) \rceil)$ levels for each input. Far level $\ell = 1, \dots, L(n) - 1$ has block size $g_\ell = r2^{\ell-1}$. K/V bases are independent across layers, levels, and heads and shared across blocks at each level. Their entries are initialized as $\mathcal{N}(0, 1)/\sqrt{g_\ell}$. Near and far levels use separate softmax operations and their outputs are summed.

Training. Each run uses 10,000 AdamW updates with peak learning rate 10^{-3} , 100 warmup steps, cosine decay to about 10^{-4} , weight decay 0.01, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, gradient norm clipping at 1, and CUDA BF16. Cross entropy is applied only at the query, with no gradient accumulation. Batches are generated online. Models within a seed receive identical training samples.

Training starts at length 64. Successive powers of two from 128 to 8192 are introduced at updates 1401, 1801, $\dots$, 3801. Each update samples uniformly from all lengths introduced

so far, with $B(n) = \min(64, \lfloor 16384/n \rfloor)$ examples. Depths are uniform on $[0, 100]$. All test lengths occur during training, so this is not length extrapolation. We evaluate update 10,000, not a validation selected checkpoint. Runs use an NVIDIA A100 80GB, Python 3.11.2, PyTorch 2.10.0+cu128, and CUDA 12.8.

Evaluation. Testing uses lengths 1024, 2048, 4096, and 8192 and depths 0%, 10%, 25%, 50%, 75%, 90%, and 100%, with 256 examples per cell and batches of four. All models and seeds share the same 7168 test examples from base seed 104729. The metric is exact match accuracy. Table 3.11 reports means and sample standard deviations across three training seeds. The six nonadjacent depths are averaged within each seed first. Runs are not pooled as independent test samples. The protocol was fixed after pilot runs required every Full seed to reach at least 90% mean accuracy across the seven depths at each test length on separate validation data. Pilots are excluded and the final test stream is separate.

Length	Nonadjacent (0–90%)		Local (100%)	
	Full	FMA ($p = 4$)	Full	FMA ($p = 4$)
1024	99.98 ± 0.04	94.36 ± 9.55	100.00 ± 0.00	100.00 ± 0.00
2048	100.00 ± 0.00	92.01 ± 14.42	100.00 ± 0.00	99.35 ± 1.13
4096	99.98 ± 0.04	81.17 ± 11.18	98.96 ± 1.80	92.81 ± 6.16
8192	98.98 ± 1.17	73.39 ± 13.07	97.66 ± 2.82	80.39 ± 7.39

Table 3.11: Controlled symbol retrieval accuracy (%). Values are mean $\pm$ sample standard deviation across seeds 0, 1, 2. Nonadjacent averages the six depths from 0% to 90%. Local places the target immediately before the query. Far field attention remains enabled in the local control.

Controls and results. Depth 100% tests local reading with far field attention still enabled. At depths 0, 50, 100%, paired controls use the first 64 examples per cell. Target replacement changes v to $(v + 1) \bmod 256$ and updates the label, leaving all distractors and positions unchanged. At length 1024 and depths 0% and 50%, all three FMA seeds achieve 100% after replacement. This supports dependence on the distant target value.

FMA’s nonadjacent accuracy falls from 94.36% at 1024 tokens to 73.39% at 8192, versus 99.98% and 98.98% for Full. Variation across seeds is substantial for FMA. Its local accuracy also falls to 80.39% at 8192. This does not establish compression as the sole cause of the gap. The experiment demonstrates access to some distant details, but not uniformly reliable retrieval across the tested lengths. It does not isolate summary capacity from training or optimization effects.

3.10 Discussion and Limitations

Fast Multipole Attention replaces dense all pairs attention by a learned multilevel operator. The method keeps exact local interactions and represents distant interactions through trainable summaries. The one dimensional version uses interval hierarchies for sequences. The two dimensional version uses quadtree cells for image grids. This gives every layer a global receptive field while avoiding materialization of the dense attention matrix.

The experiments support the main design claim. On enwik8 and WikiText-103, FMA performs better than the efficient attention baselines tested here and remains close to full attention when the dense model fits. On ImageNet-1K and ADE20K, FMA2D works as a replacement for attention in vision backbones and improves the accuracy and memory tradeoff in the tested settings. The COCO experiment further shows that its outcome depends on whether the backbone was pretrained with the same attention operator. The ablations show that the learned far field summaries matter. Uniform average pooling is worse, and increasing the rank improves accuracy at additional cost.

There are also clear limitations. First, FMA is not an exact attention algorithm. It changes the operator. This is different from FlashAttention, which computes dense attention exactly with better memory traffic. When input length is short and exact dense attention fits comfortably, an optimized exact kernel may be simpler and faster.

Second, FMA depends on a geometric hierarchy. This is natural for ordered sequences and image grids, but it is less obvious for data without a meaningful order or spatial layout. If the hierarchy groups unrelated positions, far field summaries may hide important distinctions. The method therefore fits best when distance in the chosen index space carries useful information.

Third, the hyperparameters r and p matter. A small r makes the exact near field narrow and can lose local detail. A large r increases cost. A small p may underrepresent distant groups. A large p weakens the memory saving. The retrieval test in Section 3.9 shows that $p = 4$ retains some distant details but trails Full at long contexts. Local accuracy also declines. These results do not isolate compression as the cause of the gap.

Fourth, implementation complexity is higher than for dense attention. FMA needs hierarchy construction, learned aggregation kernels, masks for near and far lists, and custom kernels to be efficient. The throughput results show that lower memory does not automatically mean faster wall clock time. Better fused kernels and IO aware tiling for the far field are important engineering work.

Finally, the approximation analysis is empirical. It shows that trained dense attention layers in a tested language model can be closely approximated by FMA summaries. It does

not give a general error bound comparable to classical Fast Multipole Method analysis for analytic kernels. Such a bound would need strong assumptions on learned query, key, and value distributions.

The same hierarchy idea suggests several extensions. Three dimensional data such as video or volumetric images could use octrees. Relative or rotary position encodings could be combined with the hierarchy for long context language models. The near field could use optimized exact attention kernels, while the far field could use more hardware aware reductions. These are extensions of the same principle rather than changes to the core idea. Local detail is kept exact, and distant context is compressed in a learned multilevel representation.

Chapter 4

Symbolic Equation Recovery and Selection

This chapter develops two complementary approaches to symbolic equation recovery. Section 4.1 presents MOSAIC-SR, which combines a pretrained symbolic prior with explicit search and repair. Section 4.2 turns to residual invariance for selecting causal symbolic models across environments.

4.1 Transformer Guided Symbolic Regression with MOSAIC-SR

This section develops MOSAIC-SR as the first symbolic recovery method of the thesis. It first presents the model and primary experiments, then examines the search dynamics through visualizations and ablations. Experiment settings and qualitative predictions are included with the corresponding evaluations. The general symbolic regression and Monte Carlo tree search background is provided in Sections 2.4 and 2.4.6.

4.1.1 Introduction

As reviewed in Section 2.4, symbolic regression seeks a closed form expression from numerical observations. Given $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$, the goal is to find an expression $e \in \mathcal{E}$, with induced function f_e , such that $y_i \approx f_e(\mathbf{x}_i)$. The regularized objective in

Equation (2.69) combines discrete structure search with continuous constant optimization. Its combinatorial structure makes exact symbolic regression computationally difficult in general [163].

Search based systems can revise operators, subtrees, variables, and constants after observing a dataset, but they usually repeat structure discovery for every new problem. Pretrained neural systems amortize part of this cost by learning equation priors from many synthetic tasks. They can generate useful candidates quickly, but a high quality proposal is not necessarily an exact recovery. A small error in a variable, operator, or constant can preserve a high test R^2 on a restricted domain while changing the recovered law.

The central question in Section 4.1 is therefore how to move from a promising neural proposal to an algebraically correct equation. For a target expression f^* and a prediction $\hat{f}$, high predictive accuracy does not imply algebraic equivalence. We therefore report both predictive accuracy, defined by test $R^2 > 1 - \varepsilon$ for some small ε , and symbolic solution rate under a fixed symbolic verification protocol. We use the same symbolic verification protocol as SRBench to determine whether a predicted expression is equivalent to the known ground truth equation [21]. Accordingly, our experiments emphasize methods that can move from a good numerical fit to an algebraically correct formula.

MOSAIC-SR addresses this question by using neural generation to guide symbolic search. A pretrained Transformer maps a small subsample of observations to multiple symbolic sketches, concentrating search in promising regions of the expression space. Monte Carlo tree search (MCTS) maps decoded variable tokens to observed inputs and produces complete equations that initialize independent local search paths. Within each path, structural edits alternate with constant refitting to correct errors in variables, operators, subexpressions, and numerical scale. Logarithmically spaced constant initializations cover scientific constants that may span many orders of magnitude.

Contributions. The contributions developed in this section are summarized as follows. First, a recovery oriented pretraining scheme samples equations, normalizes them under a fixed set of simplification rules, and evaluates them under varied input distributions, domains, and point clouds. For contrastive learning, algebraically equivalent rewrites form positive pairs, whereas structurally similar but incorrect expressions serve as hard negatives. This strategy encourages the model to generate sketches that preserve symbolic structure even when alternative expressions fit the sampled observations well.

Second, the Transformer guided recovery framework transforms decoded sketches into complete equations through parallel local search with tree based structural repair and scale aware constant fitting. This separation allows the search to correct both structural and

numerical errors and directly targets symbolic recovery rather than predictive fit alone. Like constructing a mosaic from selected pieces, MOSAIC-SR identifies and assembles symbolic components that explain the observed numerical data.

Third, the evaluation covers SRSD-Feynman with and without dummy variables and six additional benchmarks. MOSAIC-SR achieves the highest symbolic solution rate in every reported dataset while ranking among the top two methods in predictive accuracy.

4.1.2 Method

MOSAIC-SR is motivated by the observation that a pretrained model can often identify the broad form of an equation, while a nearly correct prediction may still contain the wrong variable, operator, or constant. To address this limitation, we use the pretrained Transformer to propose equation sketches and then apply search based refinement to correct structural errors.

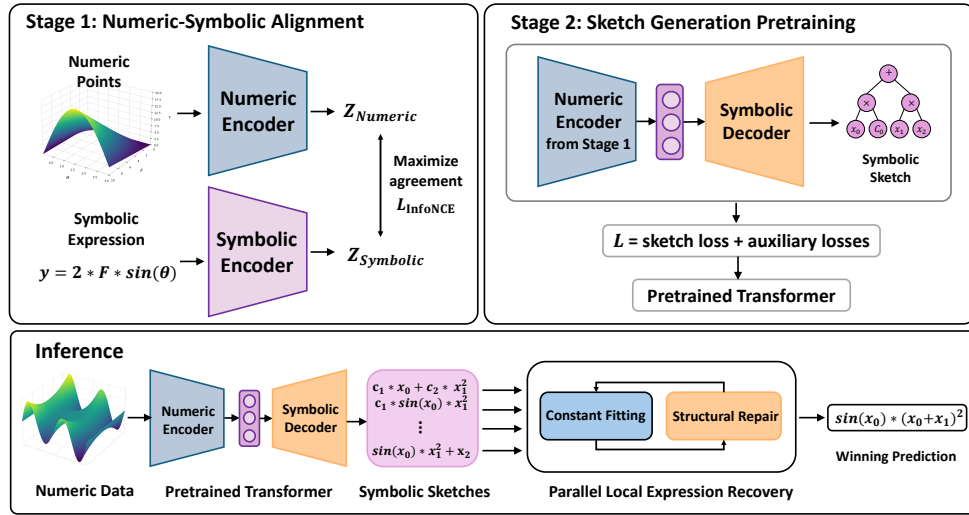


Figure 4.1: Overview of MOSAIC-SR. Stage 1 aligns numerical observations with symbolic expressions through contrastive pretraining, and Stage 2 trains the decoder to predict operator structures with variable and constant placeholders. At inference time, MCTS maps these variable tokens to observed inputs, and the resulting equations initialize parallel search trajectories that alternate constant fitting with structural repair. The final expression is selected according to validation NMSE and complexity.

Figure 4.1 summarizes the three phases of MOSAIC-SR. First, the numeric and symbolic encoders are jointly pretrained with a contrastive objective that aligns numerical observations with their corresponding expressions. The pretrained numeric encoder is then connected to an autoregressive decoder and further trained to generate symbolic sketches. Second, MCTS uses the decoded sketches and learned structural priors to assign variables and fit constants. This produces a diverse set of promising initial equations. Third, parallel local searches refine these equations through structural edits and constant optimization. The final expression is selected according to validation NMSE and complexity.

Pretraining: Learning the Symbolic Sketch Prior

Synthetic Numeric and Symbolic Pairs. The sketch prior is trained entirely with synthetic equations, following the pretraining strategy used by E2E-SR [62]. We sample expressions with one to ten variables from a predefined symbolic vocabulary. Its binary and unary operator sets are

$$\Omega_{\text{bin}} = \{+, -, \times, \div\}, \quad \Omega_{\text{unary}} = \{(\cdot)^2, (\cdot)^3, \sin, \cos, \exp, \log, \sqrt{\cdot}, \tan, \tanh, \arctan\}.$$

Each expression is represented in prefix Polish notation, where every operator appears before its operands. For example, $x_1 + \sin(x_2)$ is written as (ADD X_1 SIN X_2). In contrast to the data pipelines described for E2E-SR, we use SymPy [107] to normalize each sampled expression under a fixed set of simplification rules. For example,

$$3 + x + 2x + \sin(y \cdot x) - 1 \mapsto 2 + 3x + \sin(xy). \quad (4.1)$$

This step reduces the number of algebraically equivalent forms treated as distinct symbolic targets. We use the normalized expression to create two symbolic sequences: the full sequence and the sketch sequence. The full sequence contains the exact operators, variable identities, and constants and is used by the symbolic encoder during contrastive pretraining. The sketch sequence preserves the operator tree and leaf positions while replacing each variable and constant leaf with a corresponding special token. This sequence serves as the decoder target. We then randomly choose an input distribution and domain from predefined sets, sample 200 input points, and evaluate the expression to generate the corresponding output values. Consequently, the same symbolic expression can be observed through different input ranges rather than a single fixed set of observations.

Numeric encoder. Let $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ denote the numeric observations, with $n = 200$ during pretraining. We pad all inputs to the maximum variable dimension to maintain a

consistent input width across tasks. We then concatenate each padded input with its output and apply the following signed logarithmic transform elementwise:

$$\phi(a) = \text{sign}(a) \log(1 + |a|). \quad (4.2)$$

Since the order of observations is arbitrary, the transformed rows are encoded as a set rather than a sequence. A Set Transformer [86] contextualizes each row using information from the other observations, and attention pooling summarizes the resulting features:

$$\mathbf{M}_D = \text{SetEnc}(\{\phi(\mathbf{r}_i)\}_{i=1}^n), \quad \mathbf{z}_D = \text{Pool}(\mathbf{M}_D). \quad (4.3)$$

The set representation $\mathbf{M}_D$ is retained for decoder cross attention, while the pooled representation $\mathbf{z}_D$ is used only for numeric and symbolic contrastive learning [105].

Symbolic encoder. The symbolic encoder processes the normalized prefix sequence of an expression. In prefix notation, each operator appears before its operands, and its fixed arity determines how many subsequent subexpressions belong to it. The expression tree can therefore be parsed unambiguously without parentheses. Each token in the prefix sequence is mapped to an embedding. The resulting sequence is passed through a Transformer encoder with learned positional embeddings [160]. Masked mean pooling over the output tokens produces a symbolic embedding $\mathbf{z}_e$ for the complete expression. This encoder is used during the contrastive learning stage to teach the numeric encoder what equation generated the observations.

Stage 1: Numeric and Symbolic Alignment. We train the numeric and symbolic encoders with the bidirectional InfoNCE objective used by CLIP [125, 158]. Following SNIP, we apply this objective to align each set of numerical observations with its corresponding symbolic expression [105]. For a minibatch of B matched observation sets and expressions, define the temperature scaled similarity

$$s_{ij} = \frac{\cos(\mathbf{z}_{D_i}, \mathbf{z}_{e_j})}{\tau}. \quad (4.4)$$

The bidirectional InfoNCE loss is

$$\mathcal{L}_{\text{con}} = -\frac{1}{2B} \sum_{i=1}^B \left[\log \frac{e^{s_{ii}}}{\sum_{j=1}^B e^{s_{ij}}} + \log \frac{e^{s_{ii}}}{\sum_{j=1}^B e^{s_{ji}}} \right]. \quad (4.5)$$

This loss increases the similarity of each matched pair relative to all mismatched pairs in both directions, aligning the observations with their corresponding symbolic expressions in a shared embedding space.

Stage 2: sketch generation. After contrastive pretraining, an autoregressive Transformer decoder is attached to the numeric encoder. Training uses teacher forcing and token level cross entropy $\mathcal{L}_{\text{CE}}$ against the ground truth sketch sequence of operators, variable tokens, and constant placeholders. Several lightweight prediction heads are trained jointly to estimate coarse structural properties such as the root arity, root operator class, and expression length bucket. The second stage sketch generation objective is summarized as

$$\mathcal{L}_{\text{sketch}} = \mathcal{L}_{\text{CE}} + \lambda_{\text{aux}} \mathcal{L}_{\text{aux}}, \quad (4.6)$$

where $\mathcal{L}_{\text{aux}}$ collects the structural prediction losses and λ_{aux} controls their contribution. These auxiliary objectives encode structural information in the numeric representation, which then guides the construction of initial equations.

Inference: From Sketches to Initial Equations

The numeric encoder maps the numerical observations to latent representations. Stochastic decoding then samples diverse high probability sketches from these representations [56]. We discard duplicate sketches and those that violate the prefix grammar. Each retained sketch defines an expression skeleton with unresolved variable and constant tokens. MCTS assigns input variables and constant optimization fits the numerical values on the training set [140]. We rank the resulting complete expressions by training NMSE and use the top 24 to initialize the following independent local search trajectories.

Inference: Parallel Local Equation Recovery

Each search path maintains one current expression and attempts to improve it through a sequence of small tree edits. Different decoded sketches and MCTS assignments seed the search paths in distinct regions.

Scale Aware Constant Fitting. Every candidate tree may contain several free constants. For a fixed structure $e(\mathbf{x}; \mathbf{c})$, constants are estimated by minimizing squared error on the training set,

$$\hat{\mathbf{c}} = \arg \min_{\mathbf{c}} \sum_{(\mathbf{x}_i, y_i) \in D_{\text{train}}} (e(\mathbf{x}_i; \mathbf{c}) - y_i)^2. \quad (4.7)$$

The objective is solved with Levenberg–Marquardt optimization. The initializations are logarithmically spaced across multiple orders of magnitude. During local search, constants are refitted periodically after several structural edits.

Structural repair. The search paths run in parallel, each maintaining an independent current expression. A path iteratively modifies its expression by replacing an operator, variable, or small subtree. After several mutations, the constants are refitted and the candidate is evaluated on D_{train} using NMSE normalized by the target variance. The candidate replaces the current expression only if it achieves a lower NMSE. Each path performs 300 mutation attempts.

Inference: Selection and Simplification

Throughout local search, a shared Hall of Fame stores the best candidates found across all search paths. Once all paths have completed 300 mutation attempts, MOSAIC-SR selects the final expression from this candidate pool using D_{val} . Candidates with validation NMSE below 10^{-15} are classified as near exact fits. When one or more candidates satisfy this threshold, the selector chooses the expression with the fewest operators. Otherwise, it selects the candidate with the lowest validation NMSE. This criterion favors structurally simpler expressions once the validation error becomes near zero.

The selected expression is then simplified by replacing small constants with zero, and fitted values close to common physical constants are snapped to their symbolic forms. A physical constant snapping is retained only when its validation error remains within a small tolerance of the original expression.

4.1.3 Experiment Settings

The evaluation has two main components. The first studies scientific equation recovery on SRSD-Feynman, including its dummy variable variant. The second tests transfer to six established symbolic regression benchmarks. The results are reported in Sections 4.1.4 and 4.1.5. Later sections provide ablations on the SRSD-Feynman Easy split, search trajectory diagnostics, and qualitative examples.

Evaluation Protocol

We evaluate numerical accuracy and symbolic solution rate following the SRBench protocol. For every run performed for this chapter, the test set is used only for final scoring and the returned expression is independently parsed and reevaluated outside the method. Invalid expressions, nonfinite predictions and timeouts remain failures in the full benchmark

denominator. Each evaluation reports a numerical recovery rate and a symbolic solution rate. For a set of problems $\mathcal{P}$, symbolic solution rate is

$$\text{Sym} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \mathbf{1} [\text{SymVerify}(\hat{e}_p, e_p^*) = 1], \quad (4.8)$$

where $\hat{e}_p$ is the returned expression, e_p^* is the ground truth expression, and SymVerify is the fixed symbolic verifier. We use the SRBench symbolic solution verifier [21]. The numerical criterion is specified below in each experiment.

SRSD-Feynman Settings

Benchmark. SRSD-Feynman contains 120 equations drawn from physics with known ground truth [103]. Each equation has physically motivated sampling ranges. The benchmark is divided into Easy, Medium, and Hard splits with 30, 40, and 50 problems, respectively. Each problem provides 8000 training, 1000 validation, and 1000 test samples. We also evaluate the dummy variable variant, which adds between one and three irrelevant input columns at random positions. Results for both the original and dummy variable variants are reported in Section 4.1.4. Additional baselines used only in this evaluation are EPLEX [80], DGSR [55], TaylorGP [50], DySymNet [91], FFX [104], and BSR [60].

For this evaluation, predictive quality is measured by

$$R_p^2 = 1 - \frac{\sum_i (y_i - f_{\hat{e}_p}(\mathbf{x}_i))^2}{\sum_i (y_i - \bar{y})^2}, \quad (4.9)$$

and Acc is the percentage of test problems with $R^2 > 0.999$.

Compared methods. We compare MOSAIC-SR with search based, neural, and hybrid symbolic regression methods. For methods reported in the original SRSD-Feynman study, we show the published values from Matsubara et al. [103]. The remaining methods are evaluated under the common pipeline used in this chapter and are shown separately in the table. These evaluations include the GP methods SBP-GP and GP-GOMEA [162, 164], the Transformer and MCTS hybrid TPSR [140], the reinforcement learning and MCTS method RSRM [177], and the pretrained Transformer model NeSymReS [14].

Implementation. MOSAIC-SR uses 200 sampled input and output pairs to generate 48 candidate sketches over three nucleus sampling passes. MCTS explores variable assignments for these sketches, and its top 24 complete expressions initialize 24 local search trajectories with 300 mutation steps each. Constants and local edits are optimized on the training data. Final selection uses validation error and expression complexity. All methods used a budget of 3,600 seconds per problem and the settings recommended by their original implementations. Results for MOSAIC-SR and all newly evaluated baselines are reported as means over five seeds. Published values are taken directly from the original SRSD-Feynman study. Because that study did not report runtime, runtime is not compared on SRSD-Feynman. Mean wall clock runtime is instead analyzed in the six cross benchmark evaluations. Experiments were run on a cluster with four 80 GB H100 GPUs, four 80 GB A100 GPUs, and CPU nodes. Each training run used a single GPU without distributed training.

Cross Benchmark Settings

We further evaluate MOSAIC-SR on six widely used symbolic regression datasets: Nguyen-12 [156], Korns-15 [74], Keijzer-15 [67], Vladislavleva-8 [165], Strogatz-14 [21, 147], and Livermore-22 [109]. We compare against three groups of baselines. The search based baselines are PySR, Operon, and GP-GOMEA [18, 29, 162]. The neural baselines are E2E-SR, SNIP-SR, and GenSR [62, 89, 105]. The hybrid baselines are TPSR and SR4MDL [140, 183].

Numerical recovery requires test NMSE below 10^{-6} . We follow the standard setting for each dataset. Nguyen uses independent uniform samples from the equation domains. Korns uses independent 10000 point fit and test samples over five inputs in $[-50, 50]$. Keijzer and Vladislavleva use their equation specific sampling methods and sample sizes. Livermore follows the equation specific domains and sample counts distributed with DSO [120]. Strogatz uses the fixed 75/25 SRBench split of the PMLB ODE datasets [21]. For benchmarks without a predefined validation set, we select the final candidate using the training data. MOSAIC-SR uses the same 24 by 300 search configuration on every dataset. All baselines use their default budgets.

Implementation Details

The key implementation settings used in the reported experiments are summarized below.

Table 4.1: Core optimization settings used in the two pretraining stages.

Setting	Stage 1	Stage 2
Batch size	128	128
Training steps	400,000	250,000
Peak learning rate	10^{-4}	10^{-5}
Encoder learning rate	10^{-4}	5×10^{-6}
Warmup steps	16,000	2,000
Learning-rate schedule	Cosine decay	Constant
AdamW β_2	0.999	0.95
Weight decay	0.01	0.01
Gradient clipping	1.0	1.0

Table 4.2: Detailed Transformer architecture used by MOSAIC-SR, including the numeric encoder, symbolic encoder, and symbolic decoder. The complete architecture has approximately 17.4 million parameters and is relatively lightweight.

Parameter	Value
<i>Set-Transformer numeric encoder</i>	
Architecture	Set Transformer
Input dimension	11 (10 variables + output)
ISAB blocks	4
Hidden dimension	256
Attention heads	8
Inducing points per ISAB	32
PMA seed vectors	16
Residual normalization	Pre-LayerNorm
PMA output dimension	$16 \times 256 = 4,096$, then projected
<i>Symbolic Transformer encoder</i>	
Encoder layers	6
Hidden dimension	256
Attention heads	8
Feed-forward dimension	1,024
Dropout	0.1
Activation	GELU
<i>Symbolic Transformer decoder</i>	
Decoder layers	4
Hidden dimension	256
Attention heads	8
Feed-forward dimension	1,024
Dropout	0.1
Maximum output length	128 tokens
<i>Shared settings</i>	
Vocabulary size	45
Maximum variables	10
Total parameters	Approximately 17.4 M

Table 4.3: MOSAIC-SR inference time decoder, MCTS, and local-search budget for the cross-benchmark experiments.

Parameter	Value
Independent search paths	24
Mutations per path	300
Decoder/search passes	6
Decoder sketches retained	24
Workers	12
Conditional MCTS	Enabled
MCTS trigger / expansions	Validation NMSE 10^{-4} / 1,500
Maximum evolved AST size	32
Constant-fitting starts	16

Table 4.4: SRSD-Feynman benchmark groups, variable count ranges, dummy variable distribution, data split sizes, and input domain extrema [103]. The extrema summarize equation specific sampling distributions rather than a shared interval. Dummy counts give the numbers of problems containing one, two, or three irrelevant variables.

Group	Problems	Variables	1/2/3 dummy	Train/Val/Test
<i>Without dummy variables</i>			Input domain extrema: 10^{-30} – 10^4	
Easy	30	1–5	0/0/0	8,000/1,000/1,000
Medium	40	1–6	0/0/0	8,000/1,000/1,000
Hard	50	1–8	0/0/0	8,000/1,000/1,000
<i>With dummy variables</i>			Input domain extrema: 10^{-33} – 10^{33}	
Easy + Dummy	30	3–8	6/11/13	8,000/1,000/1,000
Medium + Dummy	40	3–8	12/18/10	8,000/1,000/1,000
Hard + Dummy	50	3–10	13/18/19	8,000/1,000/1,000

Table 4.5: Numbers of problems, variable count ranges, training and test sample counts for the Nguyen [156], Korns [74], Keijzer [67], Vladislavleva [165], Strogatz [21, 147], and Livermore [109] benchmarks.

Dataset	Problems	Variables	Training rows	Test rows
Nguyen	12	1–2	20	20
Korns	15	5	10,000	10,000
Keijzer	15	1–3	20–1,000	120–361,201
Vladislavleva	8	1–5	30–1,024	221–93,636
Strogatz	14	2	300	100
Livermore	22	1–2	20–1,000	20–1,000

4.1.4 SRSD-Feynman Results

Table 4.6 reports the SRSD-Feynman results. Numerical accuracy and symbolic recovery can differ substantially. Some baselines obtain high R^2 based accuracy but low symbolic solution rates. For example, SBP-GP reaches 87.3% accuracy on the Easy split but has an 8.0% symbolic solution rate. MOSAIC-SR reduces this gap, achieving the highest symbolic solution rate on every difficulty split and ranking among the top two methods in predictive accuracy.

Table 4.7 shows that dummy variables make symbolic recovery substantially harder for many baselines. For example, the Easy symbolic solution rate of PySR falls from 60.0% in the clean setting to 20.0%, and that of DGSR falls from 36.7% to 3.3%. MOSAIC-SR retains symbolic solution rates of 74.0%, 54.5%, and 31.2% on the Easy, Medium, and Hard splits, respectively. These rates exceed the strongest baseline by 34.0, 18.5, and 14.0 percentage points. Compared with the clean setting, they decrease by only 10.7, 0.5, and 3.6 percentage points. Predictive accuracy is also best on Easy, tied for best on Medium,

Table 4.6: Results on SRSD-Feynman. Easy, Medium, and Hard contain 30, 40, and 50 equations. Published baseline values are taken from the original SRSD-Feynman study. New evaluations performed for this chapter are means over five seeds. Acc is the percentage of equations with $R^2 > 0.999$, and Sym is the symbolic solution rate in percent. The top three values per column, including ties, are bolded. The best is also italicized.

Method	Type	Easy (30)		Medium (40)		Hard (50)	
		Acc	Sym	Acc	Sym	Acc	Sym
<i>Cited from SRSD-Feynman</i>							
uDSR	RL	100.0	50.0	75.0	17.5	20.0	4.0
PySR	GP	66.7	60.0	45.0	30.0	38.0	4.0
DSR	RL	63.3	46.7	45.0	10.0	28.0	2.0
AI Feynman	Rules	33.3	30.0	5.0	2.5	6.0	2.0
E2E	Neural	26.7	0.0	17.5	0.0	14.0	0.0
AFP-FE	GP	26.7	23.3	2.5	2.5	4.0	0.0
AFP	GP	20.0	20.0	2.5	2.5	4.0	0.0
gplearn	GP	6.7	6.7	7.5	0.0	2.0	0.0
<i>Evaluated for this chapter</i>							
SBP-GP	GP	87.3	8.0	82.0	4.0	56.0	0.0
SR4MDL	Hybrid	86.7	63.3	66.0	43.0	48.4	21.2
TPSR	Hybrid	86.0	47.3	81.0	44.0	52.0	20.0
NGGP	Hybrid	65.3	36.7	52.0	9.0	34.8	0.4
RSRM	Hybrid	64.0	0.0	47.0	0.0	22.8	0.0
GP-GOMEA	GP	63.3	24.7	65.0	19.5	44.0	10.0
EPLEX	GP	62.0	6.7	55.0	10.0	28.0	4.4
DGSR	Neural	58.7	36.7	49.0	11.5	22.4	0.0
TaylorGP	GP	35.3	23.3	24.0	18.5	10.4	6.4
DySymNet	Neural	29.3	7.3	17.5	0.0	4.8	0.0
FFX	Linear	26.7	14.0	31.0	17.5	16.0	11.6
Operon	GP	23.3	2.0	32.0	7.0	14.0	6.0
NeSymReS	Neural	21.3	16.7	14.0	2.5	4.8	3.6
BSR	Bayes	0.7	0.0	0.0	0.0	0.0	0.0
MOSAIC-SR	Hybrid	96.7	84.7	90.0	55.0	53.2	34.8

and within 0.8 percentage points of the best result on Hard. These results indicate that multiple sketch initialization and structural repair preserve recovery performance when the observations contain irrelevant inputs.

Visualizations

The figures below visualize predictive accuracy and symbolic recovery. They also compare method ranks against expression complexity. The corresponding SRSD-Feynman results are reported in Section 4.1.4.

Table 4.7: Results on SRSD-Feynman with dummy variables. Published baseline values are taken from the original SRSD-Feynman study. New evaluations performed for this chapter are means over five seeds. Acc is the percentage of problems with test $R^2 > 0.999$, and Sym is symbolic solution rate in percent. The top three values in each column, including ties, are bolded. The best is also italicized.

Method	Type	Easy (30)		Medium (40)		Hard (50)	
		Acc	Sym	Acc	Sym	Acc	Sym
<i>Cited from SRSD-Feynman</i>							
uDSR	RL	53.3	10.0	37.5	7.5	12.0	0.0
PySR	GP	20.0	20.0	10.0	5.0	2.0	0.0
DSR	RL	76.7	10.0	45.0	0.0	22.0	2.0
AI Feynman	Rules	6.7	0.0	0.0	0.0	0.0	0.0
E2E	Neural	16.7	0.0	12.5	0.0	10.0	0.0
AFP-FE	GP	16.7	16.7	0.0	0.0	4.0	0.0
AFP	GP	20.0	16.7	5.0	0.0	4.0	0.0
gplearn	GP	0.0	0.0	0.0	0.0	0.0	0.0
<i>Evaluated for this chapter</i>							
SBP-GP	GP	86.7	0.0	82.5	0.0	53.2	0.0
SR4MDL	Hybrid	80.0	40.0	28.5	14.0	24.8	4.8
TPSR	Hybrid	84.0	32.0	82.0	36.0	48.0	17.2
NGGP	Hybrid	60.0	8.7	50.5	6.0	33.2	0.0
RSRM	Hybrid	60.7	0.0	45.0	0.0	22.0	0.0
GP-GOMEA	GP	60.0	4.7	62.5	9.5	42.0	8.8
EPLEX	GP	61.3	2.7	54.5	7.0	24.4	2.8
DGSR	Neural	30.7	3.3	31.5	1.5	14.0	0.0
TaylorGP	GP	26.0	6.7	33.0	10.5	14.0	6.8
DySymNet	Neural	5.3	4.7	4.5	0.0	2.0	0.0
FFX	Linear	26.7	11.3	31.5	12.5	16.0	8.8
Operon	GP	23.3	2.7	32.0	6.0	14.0	6.4
NeSymReS	Neural	8.0	4.0	6.0	0.0	2.0	0.8
BSR	Bayes	0.0	0.0	0.0	0.0	0.0	0.0
MOSAIC-SR	Hybrid	<i>94.0</i>	<i>74.0</i>	<i>82.5</i>	<i>54.5</i>	<i>52.4</i>	<i>31.2</i>

Without Dummy Variables. Figure 4.2 compares predictive accuracy and symbolic solution rate without dummy variables. Results are separated into the Easy, Medium, and Hard splits.

MOSAIC-SR achieves the highest symbolic solution rate on all three splits while remaining among the most accurate methods. Its symbolic advantage is especially clear on the Easy split and remains substantial as equation difficulty increases.

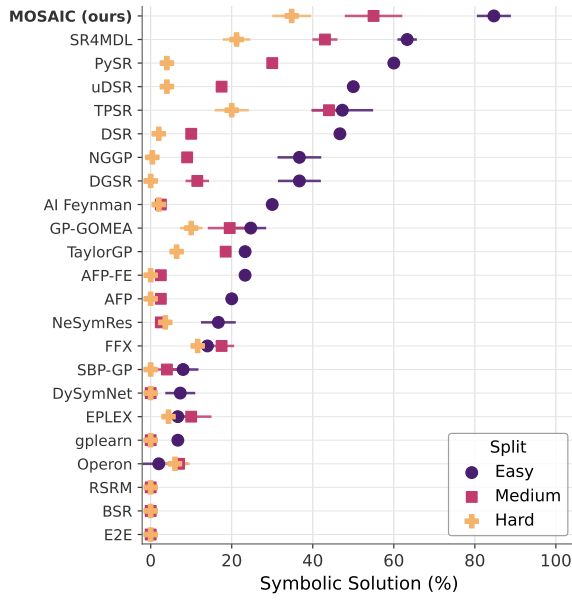
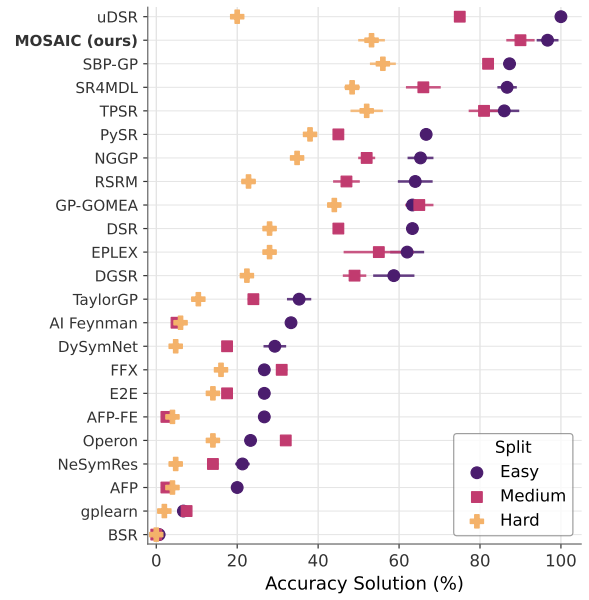
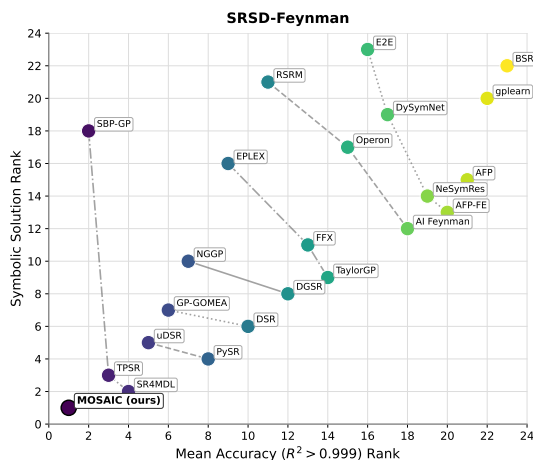
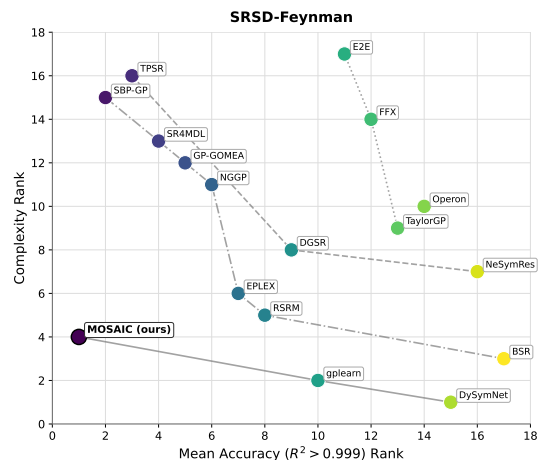


Figure 4.2: Predictive accuracy and symbolic solution rate on SRSD-Feynman without dummy variables. Points show mean performance on the Easy, Medium, and Hard splits. Horizontal bars show the reported variation across seeds.



(a) Accuracy vs. symbolic recovery



(b) Accuracy vs. complexity

Figure 4.3: Rank based comparisons on SRSD-Feynman without dummy variables. The panels compare mean predictive accuracy rank with symbolic solution rank and expression complexity rank, respectively. Methods closer to the lower left corner achieve better tradeoffs.

Figure 4.3 summarizes the rank based tradeoffs without dummy variables. Lower ranks are better in both panels. MOSAIC-SR ranks first in both predictive accuracy and symbolic recovery, giving the strongest joint result in Figure 4.3(a). It also achieves top four complexity rank in Figure 4.3(b). Methods with slightly simpler expressions have lower accuracy ranks.

With Dummy Variables. The dummy variable setting adds irrelevant input columns and therefore tests robustness to variable selection. Figure 4.4 compares predictive and symbolic performance under this perturbation.

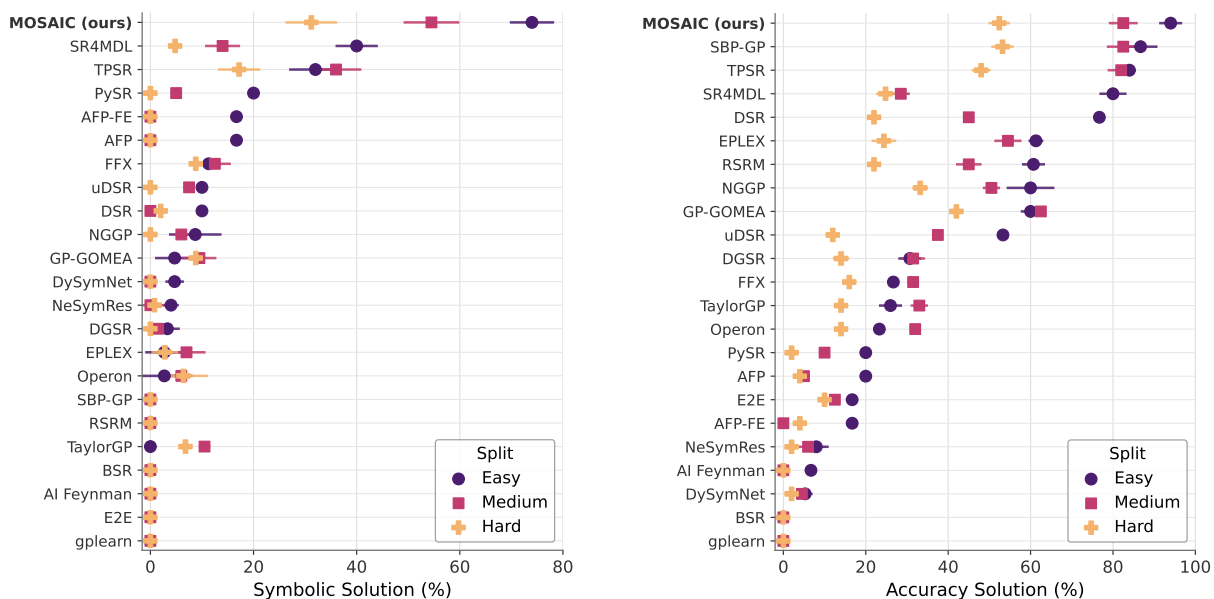
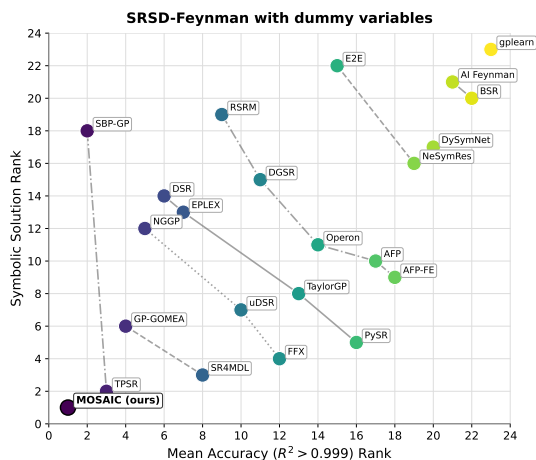
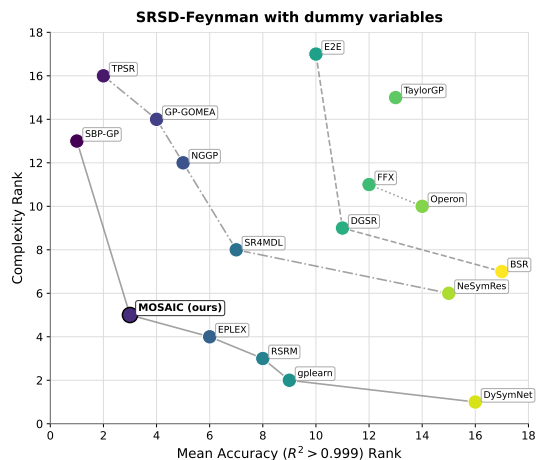


Figure 4.4: Predictive accuracy and symbolic solution rate on SRSD-Feynman with dummy variables. Points show mean performance on the Easy, Medium, and Hard splits. Horizontal bars show the reported variation across seeds.

MOSAIC-SR continues to achieve the highest symbolic solution rate on all three difficulty splits. Although irrelevant variables reduce absolute recovery on some splits, the ordering of the leading methods is preserved. Predictive accuracy also remains highest or close to highest. Figure 4.5 presents the corresponding rank based comparisons.



(a) Accuracy vs. symbolic recovery



(b) Accuracy vs. complexity

Figure 4.5: Rank based comparisons on SRSD-Feynman with dummy variables. The panels compare mean predictive accuracy rank with symbolic solution rank and expression complexity rank, respectively. Methods closer to the lower left corner achieve better trade-offs.

With dummy variables, MOSAIC-SR remains first in the joint accuracy and symbolic comparison. Its complexity rank is fifth while its accuracy rank remains near the top. This shows that robustness to irrelevant inputs does not require a large increase in recovered expression complexity.

Selected Predictions

This subsection lists predictions for twelve selected SRSD-Feynman equations, with four structurally varied examples from each difficulty split. Each table includes MOSAIC-SR and eight baselines evaluated with the common pipeline. The added cases emphasize a difference-based product and ratio, a shifted-denominator expression with a small physical constant, and a sign-sensitive multivariate product and ratio. Ellipses indicate long predictions abbreviated for readability.

Easy

Table 4.8: Predictions for feynman-i.18.16. Ground truth: $x_0 \cdot x_1 \cdot x_2 \cdot \sin(x_3)$.

Method	Prediction
MOSAIC-SR	$x_0 \cdot x_1 \cdot x_2 \cdot \sin(x_3)$
PySR	$x_0 \cdot x_1 \cdot x_2 \cdot ((x_2 - (x_3 + (x_3 + \sin(x_2) + \cos(x_3)) \cdot \cos(x_1))) \cdot (-1.090031 \cdot 10^{-8}) + \sin(x_3))$
Operon	$1.0 \cdot \sin(1.0 \cdot x_3) \cdot (-0.0027) \cdot x_1 \cdot (0.0 \cdot x_2 \cdot e^{\sin(\cos(-2.0643 \cdot x_3))} - 2.8718 \cdot x_2) \cdot (-1.1801 \cdot \dots \cdot x_1 + 127.7102 \cdot x_0) + 0.0$
GP-GOMEA	$0.560146 \cdot x_0 \cdot x_1 \cdot x_2 \cdot e^{\cos(\sin(45.244))} \cdot \sin(x_3) + 0.0$
E2E-SR	$(-0.00047451032853731212 \cdot x_1 - 19.400000000000002 \cdot (\dots - 0.0034) \cdot (30.38912823447775 \cdot x_0 + 28.70121528936986)^2) \cdot (\sin(0.5475682868652938 \cdot x_3 - 1.7094145799484706) - 0.0010999999999999998)$
SNIP-SR	$(2.9242714060991837 \cdot x_0 + 2.8138086776705413 \cdot x_2) \cdot (0.058021734319863794 \cdot x_1 - (-0.11031702200615587) \cdot 1 \cdot \dots - 1.980797909 - \frac{1}{0.6172368400935637}) - 0.3353196824463989$
GenSR	$(\dots + 24.091230291278364 \cdot x_2) \cdot \sin(6.886657042617201 \cdot 10^{-6} - 0.008195986325795546 \cdot x_1) \cdot \sin(0.0002804611337327905 - 1.0000588864638429 \cdot x_3) + 4.467432657398108 \cdot 10^{-5}$
TPSR	$(\dots + 1.3569011777772897) \cdot (0.5475682868652938 \cdot x_3 - 1.7035845799484706) \cdot (-0.005572004061167991 \cdot (0.3736301799506395 \cdot x_1 - 0.8729182770047053) - 0.004869532050182299)$
SR4MDL	$0.45952 \cdot x_3 - 0.415565 \cdot \sqrt{x_4} \cdot (x_2 + x_3) - 2.872366 \cdot \sqrt{x_4} - 7.7 \cdot 10^{-5} \cdot e^{x_1+x_3} + 3.096114 + 3.096114 \cdot 1^3 + \frac{0.000134 \cdot e^{x_1+x_3}}{\sqrt{x_4}}$

Table 4.9: Predictions for feynman-i.47.23. Ground truth: $\sqrt{\frac{x_0 \cdot x_1}{x_2}}$.

Method	Prediction
MOSAIC-SR	$1.0 \cdot \sqrt{\frac{x_0 \cdot x_1}{x_2}}$
PySR	$\sqrt{\frac{x_0 \cdot x_1}{x_2}}$
Operon	$0.0 + 1.0 \cdot (-0.3697 \cdot x_0 + 37267.543 \cdot x_1 - 0.3693 \cdot x_2 + \log(-(-1) \cdot 405434.875 \cdot x_1 + 0.0036 \cdot x_2 + 1.3023) + 0.2423) \cdot (-1.5741) \cdot \frac{1}{\dots - 1.316}$
GP-GOMEA	$0.088311 \cdot \sqrt{x_0 \cdot x_1} \cdot (\frac{x_2}{451.691759865387} + \sin(x_2) - 3.832) \cdot (-x_1 + x_2 - 8.635 + 3.845) - 2.6 \cdot 10^{-5}$
E2E-SR	$(0.3 + \frac{0.3}{0.35927349724435038 \cdot x_2 + 0.1556208603753693}) \cdot (\dots + 0.00019999999999999998 \cdot \sqrt{(77.563144512331082 \cdot x_0 - 25.889254950623197) \cdot (369968.1560669523 \cdot x_1 - 0.683865090989673)})$
SNIP-SR	$0.0008867575457021422 \cdot x_0 - (0.007603756123524418 \cdot x_1 - 0.013302360696511688 \cdot x_2 - (\dots - 0.0017) + 0.0617609181319026 + 0.07176091798886611) + 0.0042390819407224665$
GenSR	$1 \cdot \frac{1}{0.6409018128159457 \cdot x_2 + 44.900583821399} \cdot (-0.04880077057906449 \cdot x_2 - (-3.602251682762495 \cdot 10^{-5}) \cdot 1 \cdot \dots - 0.40036479485058873 \cdot x_1 - 60.5) + 0.004570229722831564$
TPSR	$0.0020274189653356795 \cdot \sqrt{\dots + 0.01000005284616329} - 8.327178391121705 \cdot 10^{-5}$
SR4MDL	$\frac{(x_1 \cdot x_1)^{0.25} \cdot (\sqrt{x_2})^{0.02439}}{\sqrt{x_3} \cdot (1 \cdot \frac{1}{x_2})^{0.04878} \cdot ((1 \cdot \frac{1}{x_2})^3)^{0.146341}}$

Table 4.10: Predictions for feynman-iii.15.27. Ground truth: $\frac{2 \cdot \pi \cdot x_0}{x_1 \cdot x_2}$.

Method	Prediction
MOSAIC-SR	$\frac{0.202642367284676 \cdot \pi^3 \cdot x_0}{x_1 \cdot x_2}$
PySR	$x_0 \cdot 6.2831855 \cdot \frac{1}{x_1 \cdot x_2}$
Operon	$1.0 \cdot (-1.72) \cdot x_0 \cdot \frac{1}{(-1) \cdot 0.5005 \cdot x_1} \cdot \frac{1}{0.5469 \cdot x_2} + 354.8659$
GP-GOMEA	$10326321737.098848 \cdot (x_0 + x_0 - (x_0 + x_1) \cdot \sin(x_0) - (-x_1 + x_2)) \cdot e^{-\sqrt{x_1} + \cos(x_1)} - 715320458.443097$

Table 4.10: Predictions for feynman-iii.15.27. (continued).

Method	Prediction
E2E-SR	$-0.00046093254802904855 \cdot x_0 + (23.8 - 0.03485279359938367 \cdot e^{0.03610003915333401 \cdot x_1}) \cdot (0.016461876715323162 \cdot x_0 + 0.004137440901804627) \cdot (\dots - 3165117.571990305 \cdot x_2 - 0.03078025829268)$
SNIP-SR	$x_0 \cdot e^{17.656406591853308 - 0.29635039365231586 \cdot x_1} \cdot 1 \cdot \frac{1}{5611389.50330818 \cdot x_2 + 6.00837704285586 \cdot 10^{-5}} - 607824857.6181431$
GenSR	$x_0 \cdot (-0.3 \cdot x_3 - 23.291253293905406 \cdot e^{-27.183755788685495 \cdot x_2} \cdot (-22.69773947774609 \cdot x_1 - 10553.262925741388)^2 + 20.414260546615335) \cdot 1 \cdot \frac{1}{\dots + 0.0} + 0.4$
TPSR	$0.021329257679301977 \cdot (0.016461876715323162 \cdot x_0 + 0.06263744090180463) - (0.030151382790591357 \cdot (\dots - 1.739119386211866 + 0.0) + 0.020088151245643712) + 0.01991184875435629$
SR4MDL	$4.5 \cdot 10^{-5} + 6.283185 \cdot 1 \cdot \frac{1}{x_3} \cdot (\frac{x_1}{x_2} + 2) - 12.566371 \cdot 1 \cdot \frac{1}{x_3}$

Table 4.11: Predictions for feynman-ii.2.42. Ground truth: $\frac{x_0 \cdot x_3 \cdot (x_1 - x_2)}{x_4}$.

Method	Prediction
MOSAIC-SR	$\frac{x_3 \cdot (x_1 - 1.0 \cdot x_2) \cdot (0.0680772584792056 \cdot \pi \cdot x_0 + 0.786128984885173 \cdot x_0 + 1.49927454094259 \cdot 10^{-46})}{x_4}$
PySR	$-1.1901392 \cdot 0.84023786 \cdot x_0 \cdot x_3 \cdot (-x_1 + x_2)$
Operon	$0.0 + 1.0 \cdot (0.0 \cdot x_1 + \frac{x_4 \cdot (-0.0018 \cdot x_1 + 0.0018 \cdot x_2) \cdot (-1.1077) \cdot x_0 \cdot (30998.9414 \cdot x_3 + \dots)}{63.3968 \cdot x_4})$
GP-GOMEA	$-0.338812 - 0.204571 \cdot \sqrt{x_3} \cdot x_0^2 \cdot (x_4 \cdot x_1 - (x_1 - x_2)) \cdot e^{\sin x_2 - x_4 \cdot x_0}$
E2E-SR	$(\dots + 0.006059999999999999) \cdot 14.7 \cdot \dots \cdot (-6.38) \cdot (0.0003498200466886168 \cdot x_0 + 0.0001793022405506989)^2$
SNIP-SR	$0.2680922469550011 + \frac{(\dots - 2.0815424010112733 \cdot x_2 + 2.07626 \cdot x_1)}{(0.752254473241603 - 0.7920820084794776 \cdot x_4) \cdot (21.622118920174636 \cdot x_4 - 0.02949544189126491)}$
GenSR	$1.0898330073263349 \cdot 10^{-7} + 0.2901744163088101 \cdot x_0 + \dots \cdot -15.679361999673807 \cdot x_3 + \dots \cdot \frac{x_1 - x_2}{x_4}$
TPSR	$0.0021005245315504836 \cdot \dots + x_0 \cdot (x_1 - x_2) \cdot x_3 \cdot x_4^{-1} $
SR4MDL	$-7002.54972 - 3506.48298 \cdot (x_4 - 2) + 0.043629 \cdot x_2 + \frac{2.398534 \cdot x_4 \cdot x_2}{\pi - \sqrt{x_1}} + 9.317983 \cdot x_5 - 0.071297 \cdot x_3 - 2.43503 \cdot x_1$

Medium

Table 4.12: Predictions for feynman-i.8.14. Ground truth: $\sqrt{(x_0 - x_1)^2 + (x_2 - x_3)^2}$.

Method	Prediction
MOSAIC-SR	$1.0 \cdot \sqrt{(-x_0 + x_1)^2 + 1.0 \cdot (x_2 - x_3)^2}$
PySR	$\sqrt{(x_0 - x_1)(x_0 - x_1 - 2.5 \cdot 10^{-7}) + (x_2 - x_3)^2}$
Operon	$1.0 \cdot (\dots + 0.1434 \cdot x_0 \cdot (-0.7572) \cdot x_1 + 0.759 \cdot x_0 - 0.1087 \cdot x_2 \cdot (-0.6665) \cdot x_2 + (0.0003 \cdot x_2 - 0.0002 \cdot x_3) \cdot (-347.5512) \cdot x_3) + 0.0$
GP-GOMEA	$0.770164 \cdot (\sqrt{x_2 - x_3} \cdot \sqrt{x_2 - x_3} + \sqrt{(-x_0 + x_1) \cdot (-x_0 + x_1)}) + 0.315539$
E2E-SR	$0.07540000000000001 \cdot \dots + (-1.272562287287044 \cdot x_0 - 0.0780069168581643) \cdot (0.00078562116483634423 \cdot x_3 + 0.00902697873784341) + 3.52 \cdot 0.25922662262108978 \cdot x_3 + 0.03979767739670666 $
SNIP-SR	$0.17291808796873243 \cdot 4.53620345462416 \cdot x_2 - 4.550165321708825 \cdot x_3 + 0.10160093303248278 \cdot \dots - 7.43758068232461 \cdot x_0 + 7.587149571342 + 0.237128160071409$
GenSR	$0.008509295312002906 \cdot 92.18053844654341 \cdot x_2 - 92.46415995095859 \cdot x_3 + 0.011009856991161837 \cdot \dots + 70.01558357608526 \cdot x_1 - 68.6354 + 0.2371299728631482$
TPSR	$0.016238973336089528 \cdot \dots + 15.111997986919254 \cdot (0.30056526331833355 \cdot x_2 + 0.02710452923829583) - 1 + 0.3188695299052845 + 0.32086952990527934 + 0.22061971100783573$
SR4MDL	$2.381338 \cdot \sqrt{x_1} + 0.086667 \cdot x_2^2 + 0.082365 \cdot x_3 \cdot (x_3 - x_4) + 0.07982 \cdot x_4^2 - 0.259968 \cdot (x_1 + x_1) + 1.351056$

Table 4.13: Predictions for feynman-bonus.18. Ground truth: $\frac{5618566741.08146 \cdot (\frac{8.98755178736818 \cdot 10^{16} \cdot x_0 + x_2^2}{x_1^2})}{\pi}$.

Method	Prediction
MOSAIC-SR	$\frac{1.60737451181046 \cdot 10^{26} \cdot x_0}{x_1^2} + 1788445339.91451 \cdot x_2^2 - 91.6922637019193$
PySR	$(x_0 + x_2 \cdot x_2) \cdot 1828542300.0$
Operon	$1.0 \cdot (-192.0 \cdot x_0 + 904458403840.0 \cdot (\dots + 247344122675455525388288.0 \cdot x_2)) - 2901952.75$
GP-GOMEA	$0.551209 \cdot (-\dots \cdot x_0 + x_2 \cdot x_2 \cdot (x_0 \cdot x_2 - 3393026213.036) \cdot \sin(11963926169.822001)) + 51687513017.491699$
E2E-SR	$(9.074947371606654 - 3.9595126882746026 \cdot 10^{-11} \cdot x_1) \cdot (0.00019487679974378762 \cdot x_0 + 89.997268492792118) \cdot (2.6396751255164016 \cdot 10^{-11} \cdot x_1 + 89.950035085595564) \cdot 0.8 \cdot (\dots - 0.0 + 55.505828465024757)$
SNIP-SR	$1.6002061046745069 \cdot (7.741215005617892 - 85.30817533148172 \cdot x_2)^2 - 0.09952661536012392 \cdot (\dots - 9.581409582884245 \cdot x_0 + 2279.3307399561345) + 5962.31522397963$
GenSR	$(74.03999172190585 \cdot x_0 + 93.5809269142206 \cdot x_2 \cdot x_2) \cdot (0.07 \cdot x_3 + 266.6340257390697)^3 + 0.4$
TPSR	$(89.04012980501867 - 0.4282907493335717 \cdot (0.00324794666239646 \cdot x_0 - 0.045525120131368176)) \cdot (89.56291606842636 - 0.9738208125647104 \cdot (\dots - 0.8327485734072698 + 4.3994585))$
SR4MDL	$\frac{927107321784793344 \cdot x_1}{x_2} + 1790558743.686736 \cdot x_3^2 - 583105027.94195 \cdot x_3 + 17382482170.32989$

Table 4.14: Predictions for feynman-i.13.4. Ground truth: $0.5 \cdot x_0 \cdot (x_1^2 + x_2^2 + x_3^2)$.

Method	Prediction
MOSAIC-SR	$0.5 \cdot x_0 \cdot (x_1^2 + x_2^2 + x_3^2)$
PySR	$(x_2 \cdot x_2 - (x_1 \cdot (2.6849122 \cdot 10^{-8} - x_1) - x_3 \cdot x_3)) \cdot 0.594442 \cdot x_0 \cdot 0.84112495$
Operon	$1.0 \cdot (-\dots + 0.0044 \cdot x_0 \cdot 0.8345 \cdot x_2 \cdot (0.0 \cdot x_0 + 135.0603 \cdot x_2 - 0.0 \cdot x_3) + 0.002 \cdot x_3 \cdot (-8.7617) \cdot x_3 \cdot (-29.1342) \cdot x_0) + 0.0$
GP-GOMEA	$-0.466588 \cdot x_0 \cdot ((-x_1 + x_2) \cdot (x_1 - x_2) - (x_1 \cdot x_2 + x_3 \cdot x_3)) + 1.557416$
E2E-SR	$0.06000000000000005 \cdot (0.25330816576971352 \cdot x_0 + 0.04483942013849232) \cdot (0.06000000000000005 \cdot (\dots + 22.0195174746776) + 0.06000000000000005 \cdot (19.7499472505697 \cdot x_1 + 2.85561506727142)^2) $
SNIP-SR	$(0.01840704821112982 \cdot x_0 + 0.027442044563726835)^2 \cdot (9.879745390847708 \cdot x_3 - 6.194380210972868 \cdot 1 \cdot \frac{1}{\dots - 4.9351334859321385 + 0.0072800592259153}) + 13.64280530393845$
GenSR	$0.009061268976888942 \cdot (1.1636683092211803 \cdot 10^{-6} \cdot x_0 - 7.759100043386413 \cdot x_1)^2 + 0.003956082223544278 \cdot (\dots - 11.742830367782478 \cdot x_2 + 2) + 2.862096358497747 \cdot 10^{-5}$
TPSR	$0.05653942421008955 \cdot \dots + 63.984404414849074 \cdot (0.28680461689317865 \cdot x_3 - 0.019022597441139 + 0.01902259744018563)^2 + 0.009063702394502786 + 1.0173098720910598 \cdot 10^{-9}$
SR4MDL	$1.251493 \cdot \dots \cdot x_2 ^{0.268858} \cdot x_2 - x_4 ^{0.338839} \cdot x_4 + 1 ^{0.205401} \cdot x_2 - 2 \cdot x_3 + x_4 ^{0.863298} \cdot \log(x_1 + 2) ^{3.150474} \cdot \sin(\log(x_1 + 2)) $

Table 4.15: Predictions for feynman-i.43.43. Ground truth: $\frac{1.380649 \cdot 10^{-23} \cdot x_1}{x_2 \cdot (x_0 - 1)}$.

Method	Prediction
MOSAIC-SR	$\frac{1.380649 \cdot 10^{-23} \cdot x_1}{x_2 \cdot (x_0 - 1)}$
PySR	$e^{\frac{\cos(\sin(0.733933372794484 \cdot x_1))}{\cos(5.7083836 \cdot \sin x_0)}} \cdot \cos(\sin(\log x_1))$
Operon	$24.9136 + 0.9928 \cdot \frac{e^{\sin(6.2437 \cdot x_1) + 0.9405 \cdot x_1 - 10.8365}}{(\sin(4.6620 \cdot x_1) + 1) \cdot (\sqrt{0.9405 \cdot x_1 + \dots})}$
GP-GOMEA	$24.874483 - 1.5238670125519831 \cdot 10^{97} \cdot e^{(x_2 - x_0) \cdot (220.661 \cdot x_0)} \cdot (\sqrt{x_1} + \sqrt{-11502.059} + \dots)$
E2E-SR	$0.00783 \cdot \left (20.5 - \frac{0.635}{(0.01955351496819132 \cdot x_0 - 0.0195847291063135) \cdot (1.49301283178395 \cdot 10^{19} \cdot x_2 - 0.0018505074099962075)}) \cdot \dots \right $
SNIP-SR	$-12.202334840601555 - 0.0035234845524787085 \cdot \frac{(5.868107132362321 - 6.00000000000062 \cdot x_2)^2}{-1.4130094715317487 + 1.4542114898814111 \cdot x_1} + \dots$

Table 4.15: Predictions for feynman-i.43.43. (continued).

Method	Prediction
GenSR	$0.5458496473257973 + \frac{60.01042781140133 \cdot \cos(17.290782890292526 - 95.0 \cdot x_2 \cdot x_3) + \dots}{-91.63629774123922 + 91.63433653923278 \cdot x_0}$
TPSR	$(0.005499999972442505 + 0.02088129903737334 \cdot \dots + 0.0003968126680374485 \cdot x_1) \cdot (\dots + x_2^{-1})$
SR4MDL	$0.000177 \cdot \frac{\frac{2}{ x_1 + 2 - \frac{4}{x_3} ^{1.000005}}}{ \log x_1 ^{0.99984} \cdot \frac{x_1}{\pi} + 2 ^{0.108164} \cdot \pi^2 ^{19.7823}}$

Hard

Table 4.16: Predictions for feynman-i.6.20. Ground truth: $\frac{\sqrt{2} \cdot e^{\frac{(-1) \cdot x_0^2}{2 \cdot x_1^2}}}{2 \cdot \sqrt{\pi} \cdot x_1}$.

Method	Prediction
MOSAIC-SR	$\frac{0.398942280401433 \cdot e^{-\frac{0.5 \cdot x_0^2}{x_1^2}}}{x_1}$
PySR	$\frac{\sin(\sin(x_0 \cdot 0.44023314 \cdot \frac{1}{x_1}) \cdot 3.1941698)}{x_0 \cdot (x_0 \cdot x_0 \cdot 0.55104643 \cdot \frac{1}{x_1 \cdot x_1} + 3.5161734)}$
Operon	$0.933 \cdot (-e^{\frac{-0.6815 \cdot x_1 - \cos(-0.9175 \cdot x_0)}{\cos(\dots)}} + \cos(-0.418 \cdot x_0)) + e^{-(0.6795 \cdot x_1 - \cos(-0.9167 \cdot x_0)) + \cos(-0.4191 \cdot x_0)} + 0.0543$
GP-GOMEA	$0.0 \cdot e^{-(\sqrt{x_0} + x_1 - 26.453) + \sqrt{-47.357} \cdot \cos(x_0)} + 0.041035$
E2E-SR	$\dots + 1.282299505658052 \cdot 10^{-8} \cdot (-0.049681955299757749 \cdot x_0 - 0.82316389700632219)^2 \cdot (3.8724298635018757 \cdot (-\operatorname{atan}(18.363084125663076 \cdot x_0 - 30.50603247242241)) - 8.06163517864935)^2 - 1.7384408$
SNIP-SR	$0.03971435283239767 + e^{1.1335725598924378 - 0.5147795206145964 \cdot x_1} \cdot 1 \cdot \frac{1}{10.003562082456531 - 0.03732680107658152 \cdot x_0}$
GenSR	$-0.0006356651178617335 + 1 \cdot \frac{1}{0.24367615501981746 \cdot x_1 + 0.0028250296256753624} \cdot 1$
TPSR	$\frac{0.537328069286272 \cdot (\dots + 38.88033280910367 \cdot \dots - 0.15054245000970234) + 8.68696568888214}{1.0926398294500483 \cdot e^{-1.4158863435762834 + 4.043613462929532 \cdot 1} \cdot \frac{1}{7.216579701937648 \cdot (\dots - 0.8885938581229681 + 0.38062016744562) + 5.539885878646895}}$
SR4MDL	$\dots + 0.003305 \cdot (\frac{2}{x_1})^2 + \frac{0.007109 \cdot (\frac{2}{x_1})^2}{e^{x_2}} - 0.058713 \cdot \sqrt{\sin(x_2)} + 0.045344 \cdot \sin(x_2) - 0.033139 \cdot \cos((\frac{2}{x_1})^2) + 0.09065 - 0.003496 \cdot 1 \cdot \frac{1}{\sin(x_2)}$

Table 4.17: Predictions for feynman-bonus.5. Ground truth: $\frac{244808.861301089 \cdot \pi \cdot x_0^{1.5}}{\sqrt{x_1 + x_2}}$.

Method	Prediction
MOSAIC-SR	$\frac{769089.720197182 \cdot x_0^2}{\sqrt{x_0 \cdot (x_1 + x_2)}}$
PySR	$769089.702160698 \cdot x_0 \cdot \sqrt{\frac{x_0}{x_1 + x_2}}$
Operon	$0.9991 \cdot (-0.0153 \cdot \dots + (-0.0018 \cdot x_0 + \frac{0.1102 \cdot x_0}{\log(-124481448181760.0 \cdot x_0 + 1.3068 \cdot x_1 + 1.3515 \cdot x_2)}) \cdot \sqrt{\sqrt{3.1423 \cdot x_0}})$
GP-GOMEA	$0.0 \cdot \sqrt{x_0} \cdot x_0 \cdot (27792532242127954164842496.0 - x_0) \cdot (-\sqrt{x_0} - 15960121508062442932928512.0 + \sqrt{x_1 + x_2})$
E2E-SR	$(9.750140310322155 - 3.5192830562550441 \cdot 10^{-10} \cdot x_0) \cdot 8.41 \cdot (-2.77 + \frac{0.04650000000000001}{-8.5274752133162408 \cdot 10^{-27} \cdot x_1 - 6.8927613619355 \cdot 10^{-27} \cdot x_2 - 0.007543467070585526})^2 \cdot (\dots + 2.4049168319)$
SNIP-SR	-
GenSR	$(0.4 \cdot x_4 + 3.617938259977237 \cdot \dots - 0.00465368) \cdot (9.8 \cdot x_5 - 104.67801968844174)^2 \cdot 185948.4313612971 - 0.0002717440950239873 \cdot x_0 - 23103053.624988906$

Table 4.17: Predictions for feynman-bonus.5. (continued).

Method	Prediction
TPSR	$-602246.9322338329 \cdot (\dots - 0.992553) - 1020047.8728000051 \cdot (3.766536263352732 \cdot 10^{-25} \cdot x_2$ $- 0.9041705958904984) + 3985390.632626627 + 3985516.888588936 + 3985756.2297664313$
SR4MDL	$\frac{769089.720197 \cdot x_1^{0.3} \cdot (x_1 \cdot x_1)^{0.6}}{\sqrt{x_2 + x_3}}$

Table 4.18: Predictions for feynman-bonus.17. Ground truth: $\frac{x_0^2 \cdot x_2^2 \cdot x_3^2 \cdot (\frac{x_3 \cdot x_4}{x_5} + 1) + x_1^2}{2 \cdot x_0}$.

Method	Prediction
MOSAIC-SR	$\frac{0.5 \cdot x_1^2}{x_0}$
PySR	$x_2 \cdot (-213.32457) \cdot \frac{1}{x_4 - 1.0.75216883}$
Operon	$1.0 \cdot (\dots + \frac{187.625}{e^{40126340.0 \cdot x_1}} + \frac{102002000.0}{e^{32823758.0 \cdot x_1}}) - 358206272.0$
GP-GOMEA	$8235444610619720859648.0 - 4117722181658013597696.0 \cdot (\sqrt{\sqrt{x_0}} + \cos(\sqrt{x_1}))$ $+ \cos(x_1 \cdot x_1 \cdot (x_2 - 64871391778.814003)))$
E2E-SR	$83.3 \cdot (\dots) \cdot (2996997476.088138 \cdot x_1 + (0.0027957553159015011 - 128292.52353780801 \cdot x_1)$ $\cdot (-886458768.06676007 \cdot x_3 - 4344590016.9748102 \cdot x_5 - 79.82124690760698) + 0.700142210497718)^2$
SNIP-SR	$2475.4420995751925 \cdot (785.1938480983365 \cdot x_1 \cdot (\dots + 501.62243037789665 \cdot (248.55715087494292 - 9.9 \cdot x_0)^3)$ $+ 456.80185093394306) + 380.8454602832127$
GenSR	$\dots \cdot (-0.23288664214316973 \cdot x_4 + 23.135639123247536 \cdot (67220856.93727316 \cdot x_1 + 0.08379184547075899)^2) - 0.2$
TPSR	$0.19718990786562984 + 90.27582145384471 - 8.75239768085726 \cdot 1$ $\cdot \frac{1}{0.8940335705346859 \cdot \sqrt{0.9471216397266307 - 0.0016772113459999217 \cdot (\dots - 0.02129158)} + 0.6302969795300203}$
SR4MDL	$0.5 \cdot \sqrt{1 \cdot \frac{1}{x_1} \cdot x_2 ^2} \cdot \sqrt{ x_4 + 1 \cdot \frac{1}{x_1} }$

Table 4.19: Predictions for feynman-iii.21.20. Ground truth: $-\frac{x_0 \cdot x_1 \cdot x_2}{x_3}$.

Method	Prediction
MOSAIC-SR	$-0.318309886183791 \cdot \pi \cdot x_1 \cdot x_2 \cdot (-1 + \frac{x_0 + x_3}{x_3})$
PySR	$\frac{x_0 \cdot x_2 \cdot (116.94989 - x_1)}{x_3}$
Operon	$-1556.0109 + 2.5895843701950685 \cdot 10^{25} \cdot \frac{2.4524}{(-2.7636805717985786 \cdot 10^{27} \cdot x_3) \cdot x_1^{-1} \cdot (-25.7193) \cdot x_2 \cdot (-0.2950) \cdot x_0}$
GP-GOMEA	$-2029461039.160109 - 2.4147765954346173 \cdot 10^{46} \cdot \cos(\cos x_1 \cdot (x_2 - 76913126293.195007)) \cdot x_1^2 \cdot (x_0 + x_3)^2 \cdot \sin x_2$
E2E-SR	$(66.829576481052452 \cdot x_2 - 0.0017371407519285152) \cdot (-92.6) \cdot (0.1078068308644714 - 3.2338505696933849 \cdot 10^{-11}$ $\cdot x_1)^2 \cdot (4.390271757193469 \cdot 10^{27} \cdot x_0 - 0.3531506873096935)^2 \cdot \dots$
SNIP-SR	$753690240.1795971 + x_2 \cdot (4.368785986955585 \cdot x_1 - 43.75053176204654 \cdot e^{17.15960871844588 - 2.0 \cdot x_0 \cdot x_3})$ $\cdot \tan(-7.954465726265568 + 73364.55500241455 \cdot \dots)$
GenSR	$-0.6 + (709.5709262937739 \cdot (0.6 \cdot x_6 + 283.85298348125826 \cdot (-76.27479590083885 + 9.0 \cdot x_5)^2) \cdot (\dots + x_0 \cdot x_1 \cdot x_2$ $\cdot x_3^{-1}))$
TPSR	$0.7537448782577802 \cdot (0.0006584109871255986 + 26.947409871392118 \cdot x_2) \cdot (0.030073288639170904$ $- 0.7428317603163052 \cdot \dots + x_0 \cdot x_1 \cdot x_3^{-1})$
SR4MDL	$189571765517.15814 + 131401134781.99579 \cdot \log 2 - 0.000427 \cdot x_4 + 8256334692.58376 \cdot \cos(x_4^{-1})$ $+ 379143531034.32043 \cdot (2 - 2 \cdot x_1) + \dots$

Table 4.20: Results and runtime on six symbolic regression benchmarks. Sym and Acc are percentages. Time is the average runtime in seconds per equation. Acc denotes test NMSE below 10^{-6} . The top three Sym and Acc values per dataset are bolded.

Method	Nguyen			Korns			Keijzer			Vladislavleva			Strogatz			Livermore		
	Sym	Acc	Time	Sym	Acc	Time	Sym	Acc	Time	Sym	Acc	Time	Sym	Acc	Time	Sym	Acc	Time
PySR	8.3	91.7	99.6	20.0	60.0	619.6	26.7	73.3	100.8	12.5	25.0	126.9	42.9	71.4	150.6	31.8	72.7	146.2
Operon	0.0	0.0	4.6	6.7	53.3	20.4	0.0	20.0	6.5	0.0	12.5	11.7	7.1	21.4	9.7	0.0	0.0	6.5
GP-GOMEA	8.3	16.7	29.2	0.0	26.7	88.9	6.7	13.3	33.3	12.5	12.5	53.2	14.3	64.3	45.9	18.2	18.2	31.4
E2E-SR	0.0	0.0	9.0	0.0	6.7	21.9	0.0	0.0	13.4	0.0	0.0	18.2	0.0	0.0	13.6	0.0	0.0	12.4
SNIP-SR	0.0	0.0	26.3	6.7	6.7	171.9	0.0	0.0	25.1	0.0	0.0	101.9	0.0	7.1	25.4	0.0	0.0	24.6
GenSR	0.0	0.0	16.0	0.0	40.0	234.4	0.0	6.7	53.7	0.0	12.5	357.3	0.0	21.4	82.6	0.0	4.5	23.8
TPSR	0.0	33.3	60.3	0.0	20.0	78.3	0.0	33.3	85.5	0.0	25.0	170.9	0.0	28.6	124.4	0.0	18.2	46.9
SR4MDL	41.7	41.7	873.5	20.0	26.7	1,966.4	20.0	26.7	1,198.3	12.5	12.5	2,218.6	50.0	64.3	874.9	22.7	27.3	1,181.8
MOSAIC-SR	75.0	91.7	75.6	40.0	53.3	288.1	53.3	66.7	71.3	25.0	37.5	91.3	64.3	71.4	99.1	45.5	90.9	83.8

4.1.5 Cross-Benchmark Results

Table 4.20 summarizes the multiple benchmark results. It reports symbolic solution rate, predictive accuracy, and mean wall clock time per equation. MOSAIC-SR obtains the highest symbolic solution rate on every dataset. Improvements over the strongest baseline range from 12.5 to 33.3 percentage points. These gains cannot be explained by predictive accuracy alone. MOSAIC-SR and PySR have identical numerical recovery on Nguyen and Strogatz, but MOSAIC-SR increases symbolic solution rates from 8.3% to 75.0% and from 42.9% to 64.3%. Although PySR has slightly higher numerical recovery on Korns and Keijzer, MOSAIC-SR approximately doubles symbolic recovery on both datasets. Moreover, the neural baselines reach at most 6.7% symbolic recovery. Operon, GP-GOMEA, and direct neural generators are generally faster, but their symbolic recovery is substantially lower. In contrast, MOSAIC-SR transfers across benchmarks and better converts accurate predictions into exact equations. Among competitive recovery baselines, MOSAIC-SR is $1.3\text{--}2.2\times$ faster than PySR and $6.8\text{--}24.3\times$ faster than SR4MDL across all six datasets. This highlights the importance of search and repair after neural generation.

Selected Predictions

Six selected equations illustrate the cross-benchmark predictions. Each table includes every method retained in the comparison of Section 4.1.5.

Table 4.21: Predictions for Nguyen-4. Ground truth: $x_1^6 + x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1$.

Method	Prediction
MOSAIC-SR	$1.0 \cdot x_1 \cdot (x_1^3 + 1) \cdot (1.0 \cdot x_1 \cdot (x_1 + 1) + 1)$
PySR	$(x_1 + 1.0) \cdot (x_1 \cdot x_1 \cdot x_1 \cdot (x_1 \cdot x_1 + 1.0) - x_1 + x_1 + x_1 - 1 \cdot 3.009093 \cdot 10^{-8})$
Operon	$0.263197660446167 \cdot \exp(3.121897220611572 \cdot x_1) - 0.250046372413635$
GP-GOMEA	0.780175498806059
E2E-SR	$(0.82722131708105334 - 0.0089588507630547431 \cdot x_1) \cdot (-0.014978078619482149 \cdot x_1 - 4.659629985505114) \cdot (0.06345852623830444 \cdot x_1 + 0.06393233734253951) \cdot (1.5553560352525597 \cdot x_1 + 0.00177689565047806) \cdot (0.0137 \cdot (-6.4184099828749757 \cdot x_1 - 0.6423212964360453)^3 - 2.63)$
SNIP-SR	$-787.69917995555147 \cdot (0.00070707667540883934 \cdot x_1 + 1)^2 + 784.8060091216709 - 289.476805583656 \cdot (73.38150650502881 \cdot x_1 - 101.72796954182427)^{-1}$
GenSR	$0.77644249309791224 \cdot x_1 \cdot \exp(1.9762297509792082 \cdot x_1) - 0.052892466261292466$
TPSR	$(1.5553560352525597 \cdot x_1 - (-0.3773368424196556 \cdot (1.5553560352525597 \cdot x_1 - 0.03842310434952195) - 0.22310537140050477) - 0.03842310434952195 + 0.22310536281517437) \cdot (((0.11785642826425316 - 0.0012268956671139082 \cdot (1.5553560352525597 \cdot x_1 - 0.03842310434952195)) \cdot ((3.295880062849909 \cdot (1.5553560352525597 \cdot x_1 - 0.03842310434952195) - 0.03672358170662421)^2 + 12.63736506039501))^2 - 2.5713685830174153) \cdot (0.0006597917724206856 \cdot (1.5553560352525597 \cdot x_1 - 0.03842310434952195) + 0.031343288513934704) + 0.2842492306014455) \cdot (1.5553560352525597 \cdot x_1 - 0.03842310434952195 + 1.5051862867904549) - (0.009429561281775448 \cdot (1.5553560352525597 \cdot x_1 - 0.03842310434952195) + 0.15816147970256972)$
SR4MDL	$0.258935 \cdot \exp(3.141592653589793 \cdot x_1) - 0.246514$

Table 4.22: Predictions for Nguyen-11. Ground truth: $x_1^{x_2}$.

Method	Prediction
MOSAIC-SR	$\exp((1.0 \cdot x_2 + 1.11259407539204 \cdot 10^{-17}) \cdot \log(x_1))$
PySR	$\exp(x_2 \cdot \log(x_1 - 7.453925 \cdot 10^{-9})) + 9.259128 \cdot 10^{-9}$
Operon	$0.4424457312568320138828227 \cdot x_1 + 0.526753962039948$
GP-GOMEA	$2.01164555549622 - 1.81258428096771 \cdot \cos(2 \cdot x_2 \cdot (x_1^2 - 1.00295400619507) \cdot (5.04862117767334 \cdot x_1 + x_2 + \cos(x_2) - 0.310173720121384)^{-1} + 0.978409945964813)$
E2E-SR	$0.62014723269920278 \cdot x_1 + (0.052757803417621763 - 0.054718873473459068 \cdot x_1) \cdot (0.11515042468674191 - 0.031915295185792375 \cdot x_2) \cdot (94.508205254545958 - 203.3004303349743 \cdot x_2) + 0.4187448946002867$
SNIP-SR	$1.22898636248581 \cdot x_1 \cdot x_2 - 1.1248163129914415 \cdot x_2 + 0.9561923734505865$
GenSR	$x_2 \cdot \sin(3.090228538225934 \cdot x_1 + 0.12294672738656137) \cdot (0.033708888164141505 - 3.899959165568685 \cdot x_1)^{-1} + 0.9810675959848522$
TPSR	$(3.1915295185792374 \cdot x_2 - (6.702556612707029 \cdot (3.6479248982306043 \cdot x_1 - 2.183853561174784) + 11.474458434775793) - 5.18045843471929 - 1.5150424686741908) \cdot (3.6479248982306043 \cdot x_1 - (5.371492648034048 - 2.5796876031538174 \cdot (3.1915295185792374 \cdot x_2 - 1.5150424686741908))) - 14.370492648199649 - 2.183853561174784) \cdot (0.006338178745524264 \cdot (3.1915295185792374 \cdot x_2 - 1.5150424686741908) + 0.16228672092950178) \cdot (0.00436698533297426 \cdot \exp((3.1915295185792374 \cdot x_2 - 1.5150424686741908 - 1.0903856097940126) \cdot 0.4431520325475648 \cdot (3.6479248982306043 \cdot x_1 - 2.183853561174784)) + 0.01042951702434242)$
SR4MDL	$0.751037 \cdot (x_1 + x_1 \cdot x_2^{-1})^{0.010698} \cdot \sin(-x_1 + (x_2 + 3.141592653589793) \cdot 1^{-1} - 1)^{1.571039} \cdot (\exp(x_1 + x_1 \cdot x_2^{-1})^{0.001281})^{-1} \cdot x_2^{3.359886} \cdot (\sin(x_2)^{3.470214})^{-1}$

Table 4.23: Predictions for Strogatz-shearflow2. Ground truth: $\left(\cos^2(x_2) + \frac{1}{10} \sin^2(x_2)\right) \sin(x_1)$.

Method	Prediction
MOSAIC-SR	$(0.9 \cdot \cos(x_2)^2 + 0.09999999999999999) \cdot \sin(x_1)$
PySR	$\sin(x_1) \cdot (\cos(\sin(x_2) \cdot 11.785839^{-1}) - 0.99600285) \cdot 250.22122$
Operon	$0.05662523819582111 \cdot x_2 + 0.017341127619147$
GP-	$0.450004816055298 \cdot \sin(x_1) \cdot \sin(2 \cdot x_2 - 4.71237802505493) + 0.550020277500153 \cdot \sin(x_1)$
GOMEA	$-7.75317403167719 \cdot 10^{-6}$
E2E-SR	$0.03790008096684952 \cdot \sin(0.09445851906577567 + 0.03960031064088332 \cdot ((0.05920507672980812 - 90.0083517060878 \cdot \tan(0.49632673547196763 \cdot x_1 + 0.22256771097376267)) \cdot (0.2089712876837641 \cdot x_2 - 0.900448620172386))^{-1}) - 0.02225101908746781$
SNIP-SR	$0.013779661168824415 \cdot x_1^2 + 0.008255847908988898 \cdot x_1 - 0.13407144869680693$
GenSR	$-(0.35205590018522714 \cdot \cos(2.0175339345367123 \cdot x_2 - 3.1461725251880917) - 1.7224017370428149 \cdot (\sin(0.010364063377607676 \cdot x_2 + 2.445867155619959)^{-1})^2 \cdot \sin(0.9977696452910493 \cdot x_1 + 3.1302681031354256) + 0.0027183801796846568$
TPSR	$(-1.8419676778775944 \cdot (0.7251641631422997 \cdot x_2 + 0.6318785594782278) - (1.3871488161946197 \cdot (0.7251641631422997 \cdot x_2 + 0.6318785594782278) - 0.5424641814306489) \cdot (-0.011063257637029162 \cdot (0.36670071625456485 \cdot x_1 + 0.20006113748695625) - 1.3363229583657414) - (1.773370534513494 \cdot (0.36670071625456485 \cdot x_1 + 0.20006113748695625) - 0.623428139816112) + 0.5334281389900255) \cdot ((0.7251641631422997 \cdot x_2 - 2.49723676249942 + 0.6318785594782278) \cdot (0.8223668625439224 - 0.043119921942055064 \cdot (0.36670071625456485 \cdot x_1 + 0.20006113748695625)) - 1.3351697306115864 \cdot \sin(2.917585780146784 \cdot \sin(1.0133095548227202 \cdot (0.7251641631422997 \cdot x_2 + 0.6318785594782278) + 0.01366990205656808)) + 0.9996592250581154) \cdot (0.07210748715109865 + 1.0791074864410393 - 0.10326251767611451 \cdot 1 \cdot (0.0788989875961674 \cdot (0.36670071625456485 \cdot x_1 + 0.20006113748695625) + 0.1676990800057751)^{-1})$
SR4MDL	$-0.042743 \cdot x_1 \cdot x_2 - 0.051972 \cdot x_1 \cdot \sin(-x_2 \cdot 2 + x_2)^{\frac{1}{2}} + 1.972781 \cdot (-x_1 \cdot \sin(-x_2 \cdot 2 + x_2)^{\frac{1}{2}} + x_1) - 0.204562 \cdot \log(x_2) - 0.02844 - 0.02844 \cdot 1^3$

Table 4.24: Predictions for Livermore-1. Ground truth: $x_1 + \sin(x_1^2) + \frac{1}{3}$.

Method	Prediction
MOSAIC-SR	$x_1 + \sin(x_1^2) + 0.3333333333333333$
PySR	$x_1 + \sin(x_1 \cdot x_1) + 0.3333334$
Operon	$0.99815390177276 \cdot x_1 + 0.385755509138107$
GP-	0.416014617587274
GOMEA	
E2E-SR	$0.99510452368622162 \cdot x_1 - 1.03 \cdot \sin(0.777 \cdot (1.1888631901860294 \cdot x_1 + 0.090269873934594466)^2 - 3.77) + 0.36129693740020308$
SNIP-SR	$-90.25067542494121 \cdot (0.005531892247014312 \cdot x_1 - 1)^2 + 90.72787811926341 + 0.0022187878284230704 \cdot (0.14762793267964414 \cdot x_1 + 1.4559888306142952)^{-1}$
GenSR	$0.998153963905744 \cdot x_1 + 0.3857558099256324$
TPSR	$5.8514992167604305 \cdot (0.17060586417335055 \cdot x_1 - 0.005171897428231887) - 0.5444981796790447 \cdot \sin((0.5916941314151313 \cdot (0.17060586417335055 \cdot x_1 - 0.005171897428231887) + 0.018303324214130782) \cdot (-60.41281200227787 \cdot (0.17060586417335055 \cdot x_1 - 0.005171897428231887) - 1.8687415763958688)) + 0.03033935030352127 + 0.3613393503035212$
SR4MDL	$1.442695 \cdot \log(2) \cdot x_1 + \sin(x_1^2) + 0.066667 - -0.133333 \cdot 2$

Table 4.25: Predictions for Livermore-11. Ground truth: $\frac{x_1^2 x_2^2}{x_1 + x_2}$.

Method	Prediction
MOSAIC-SR	$1.0 \cdot x_1^2 \cdot x_2^2 \cdot (x_1 + x_2)^{-1}$
PySR	$x_1 \cdot x_1 \cdot x_2 \cdot (1.0000085 + -\sin(\cos(x_2 \cdot x_2)) \cdot 1.2525511 \cdot 10^{-5}) \cdot (x_2 + 8.6958323 \cdot 10^{-7}) \cdot (x_1 + x_2)^{-1}$
Operon	$0.011714923195541 + 0.0004353052779163251485 \cdot \exp(-16.278768064237051 \cdot x_1 \cdot x_2) \cdot \cos(1.651700139045715 \cdot x_1) \cdot x_2^{-1}$
GP-GOMEA	$1.0 \cdot x_1^2 \cdot x_2^2 \cdot (x_1 + x_2)^{-1}$
E2E-SR	$(0.06102565887369925 - 0.8666989784169188 \cdot x_2) \cdot (5.672968018606769 - 0.16969952289121198 \cdot (-1.101556530062938 \cdot x_1 - 1.7747546914056933 \cdot x_2 - 1.3923669907018425)^{-1}) \cdot (0.08756886213383504 \cdot x_1 - 0.366604846256249) \cdot (1.7505584083402312 \cdot x_2 - 0.12325958446073971)$
SNIP-SR	$-24535.521299664357 \cdot (7.784214446548355 \cdot 10^{-5} \cdot x_1 + 1)^2 + 24534.943977652507 + 4576.952109241969 \cdot (10366.057719507926 - 10583.112972836361 \cdot x_1)^{-1}$
GenSR	$0.060303244055858976 - 3.8764530048693256 \cdot 10^{-10} \cdot \exp(-36.87562797196373 \cdot x_1 \cdot x_2) \cdot \sin(1.6592119930151334 \cdot x_2 + 1.594705827381022)$
TPSR	$-(-1.5352958873665828 \cdot 10^{-9} + 1 \cdot (-1.7143334989974734 - 88.43320312834005 \cdot (1.775147604164816 \cdot x_1 + 0.03944382780585689)) + 89.67537962342587 \cdot (1.7505584083402312 \cdot x_2 + 0.006362112212304721) - 2.344333499023583)^{-1} \cdot (0.006090218920752856 \cdot (1.775147604164816 \cdot x_1 + 0.03944382780585689) - 0.0002346013421339915) \cdot (1.775147604164816 \cdot x_1 - 0.04036802544161824 + 0.03944382780585689) \cdot (30.197815111921443 \cdot (1.7505584083402312 \cdot x_2 + 0.006362112212304721) - 0.13937886363113336) \cdot (0.7168363327318004 - 88.39345664180456 \cdot (1.7505584083402312 \cdot x_2 + 0.006362112212304721)) + 7.821765446292284 \cdot 10^{-8} \cdot (1.7505584083402312 \cdot x_2 + 0.006362112212304721) + 1.1783101145614108 \cdot 10^{-7}$
SR4MDL	$-0.910576 \cdot -x_2 \cdot \cos(3.141592653589793) \cdot x_1 \cdot (x_1 + x_2)^{-1} - 0.211742 \cdot x_2 + 0.015383 \cdot (3.141592653589793 \cdot x_2) - 0.062503$

Table 4.26: Predictions for Livermore-21. Ground truth: $\exp(-x_1^2)$.

Method	Prediction
MOSAIC-SR	$\exp(-x_1^2)$
PySR	$-x_1 + \exp(x_1 \cdot 2.732575 \cdot x_1 - 2.732575^{-1}) + x_1 \cdot x_1 \cdot x_1^{-1}$
Operon	$0.307795726557961 \cdot x_1 + 0.889393270015717$
GP-GOMEA	0.80009440459554
E2E-SR	$0.4 \cdot \cos(2.1126836069969081 \cdot x_1 - 0.03706022875549485) + 0.6$
SNIP-SR	$130.29417035635083 \cdot (1 - 0.027990232692061055 \cdot x_1)^2 - 17.620730166228215 \cdot (1 - 0.20671522387448319 \cdot x_1)^2 - 111.70304890826463$
GenSR	$-x_1 \cdot \sin(0.7057024858927093 \cdot x_1 + 0.006436287112417469) + 0.9762498782540814$
TPSR	$0.38835543110695137 \cdot \cos(0.6509206497490071 - 0.954024508269838 \cdot (2.3474262299965645 \cdot x_1 + 0.6810441902716724)) + 0.610552011368743$
SR4MDL	$1.0000000014612915 \cdot \exp(-x_1^2)$

The following sections examine validation trajectories and ablations.

4.1.6 Qualitative Recovery Across Taylor Orders

We compare baseline methods at the common truncation orders $K \in \{2, 5, 7, 10\}$. Each table first lists the Taylor truncated data generating expressions and then the corresponding predictions. Long expressions are abbreviated with ellipses.

Table 4.27: Taylor recovery for Logarithm. Ground truth: $\log(x_0 + 1)$.

Method	K	Expression
Taylor data	2	$x_0 + \frac{(-1) \cdot x_0^2}{2}$
	5	$\frac{x_0^5}{5} - \frac{x_0^4}{4} + \frac{x_0^3}{3} - \frac{x_0^2}{2} + x_0$
	7	$\frac{x_0^7}{7} - \frac{x_0^6}{6} + \frac{x_0^5}{5} - \frac{x_0^4}{4} + \frac{x_0^3}{3} - \frac{x_0^2}{2} + x_0$
	10	$\frac{x_0^{10}}{10} - \frac{x_0^9}{9} + \frac{x_0^8}{8} - \frac{x_0^7}{7} + \frac{x_0^6}{6} - \frac{x_0^5}{5} + \frac{x_0^4}{4} - \frac{x_0^3}{3} + \frac{x_0^2}{2} - x_0 + \frac{(-1) \cdot x_0^{10}}{10}$
MOSAIC-SR	2	$-0.5 \cdot x_0^2 + x_0$
	5	$x_0 \cdot (0.6 \cdot x_0^4 - 0.75 \cdot x_0^3 + x_0^2 - 1.5 \cdot x_0 + 3.0)$
	7	$\log(x_0 + 1.00006603079797)$
	10	$\log(0.999982548715611 \cdot x_0 + 1.0)$
PySR	2	$x_0 \cdot (-0.50000006) \cdot x_0 + x_0$
	5	$\log(x_0 + e^{0.16825567 \cdot x_0^4 \cdot (0.1677055 \cdot x_0^2 + x_0) \cdot \sin(x_0)})$
	7	$\log\left(\frac{0.9312916 \cdot x_0^2}{(8495.134 \cdot x_0 + 4841.4176432432) \cdot \cos(x_0 + \sin(x_0) + 0.55235815)} + x_0 + 0.9999974\right)$
	10	$\frac{x_0}{\sqrt{x_0 + e^{0.08364746 \cdot \sin(\sin(x_0 \cdot (-x_0 \cdot (0.007794200238976 - 0.01248848 \cdot x_0) + x_0)))}}$
TPSR	2	$(0.001919221591558286 - 0.06896537602514179 \cdot (3.264972382184671 \cdot x_0 + 0.027804329418150822)) \cdot (\dots - 4.45999942314764)$
	5	$(\dots + 0.1477134780683493) \cdot (0.4994556780444385 \cdot (3.5857739179080634 \cdot x_0 + 0.016444471966369778) - 0.008310635538500689)$
	7	$(0.0004599611012800237 - 0.007574319367220268 \cdot (3.675921308958977 \cdot x_0 + 0.060970365116553515)) \cdot (\dots - 9.800723047144695)$
	10	$(0.00048668399774344864 - 0.008860476303170692 \cdot (3.547005732829724 \cdot x_0 + 0.05530797638861583)) \cdot (\dots - 8.998199250504804)$
E2E-SR	2	$\left(-6.33 + \frac{7.87}{\dots}\right) \cdot (\dots)$
	5	$(-9.9 - \frac{3.0}{0.055220918335784178 \cdot x_0 + 0.0842532448682821}) \cdot (-0.021873220899239185 \cdot x_0 - 3.1127899485564804 \cdot 10^{-7})$
	7	$(-0.00015 + \frac{0.0565}{3.675921308958977 \cdot x_0 + 6.6709703651165538}) \cdot (\dots + (0.16798960381942527 \cdot x_0 + 6.0327863456858267))$
	10	$(0.00791 - \frac{0.396}{-5.536875948947199 \cdot x_0 - 9.1173357511426303}) \cdot (19.54400158789178 \cdot x_0 + 0.00174694990127322)$
Operon	2	$1.0 \cdot ((1.6375 - 2.1795 \cdot x_0) \cdot (0.2294 \cdot x_0 - 0.2865) + 0.4691) + 0.0$
	5	$1.0 \cdot (- (2.7168 - 2.0625 \cdot x_0) - (\dots + 1.3181) + e^{\cos(0.2009 \cdot x_0)} + \sin(-0.6926 \cdot x_0)) + 0.0$
	7	$1.0 \cdot (\cos(-0.3504 \cdot x_0) \cdot 0.3038 \cdot x_0 - 1.3606 \cdot x_0 \cdot (-0.4265) \cdot \frac{1}{0.5983 \cdot x_0 + 0.8338}) + 0.0$
	10	$1.0001 \cdot \log(0.9999 \cdot x_0 + e^{\sin(\cos(1.5708))}) + 0.0$

Table 4.28: Taylor recovery for Sine. Ground truth: $\sin(x_0)$.

Method	K	Expression
Taylor data	2	x_0
	5	$\frac{x_0^5}{120} - \frac{x_0^3}{6} + x_0$
	7	$\frac{x_0^7}{120} - \frac{x_0^3}{6} + x_0 + \frac{(-1) \cdot x_0^7}{5040}$
	10	$\frac{x_0^9}{362880} - \frac{x_0^7}{5040} + \frac{x_0^5}{120} - \frac{x_0^3}{6} + x_0$
MOSAIC-SR	2	x_0
	5	$-0.15 \cdot x_0^3 - 0.008333333333333333 \cdot x_0 \cdot (-(x_0^2 - 1)^2 - 0.949359216081706) + 0.983755339865986 \cdot x_0$
	7	$0.995212385036962 \cdot \sin(x_0)$
	10	$\sin(0.999224487053661 \cdot x_0)$
PySR	2	x_0
	5	$\sin(x_0 \cdot (\cos(\sin(x_0 \cdot x_0 \cdot 0.099885456 \cdot \cos(\sin(x_0 \cdot (-0.6655867)))))) - 1 \cdot 0.0023687228)) \cdot 1.0040035$
	7	$\sin(\frac{x_0}{\cos(x_0 \cdot (-0.03543025) \cdot (-0.61757547 \cdot x_0 + \sin(\sin(\sin(\sin(\sin(\sin(\sin(\sin(x_0))))))))))})$
	10	$\sin(x_0 \cdot \cos(x_0 \cdot 0.0016573971 \cdot \frac{1}{(-1) \cdot 2.5042472 \cdot \frac{1}{-x_0 \cdot x_0 \cdot x_0 + \sin((x_0 + x_0) \cdot (-2.066677))}}))$
TPSR	2	$1.6952640543497626 \cdot (0.5898786076624276 \cdot x_0 - 0.1503609060667909) + 0.2549014392344917$
	5	$-(\dots + 0.008829876210206134) + 0.5311733478647308 \cdot (0.5442460160982873 \cdot x_0 + 0.03327453788862279) - 0.008829876210206504$
	7	$0.0037014237986746907 - 1.1106297415087139 \cdot \sin(6.201124685239097 - 1.7856942534231586 \cdot (\dots - 0.04590158661878021))$
	10	$0.9998616696023864 \cdot \sin(1.7327776217284636 \cdot (\dots - 0.06370211051208034) + 0.11012377319587009) + 6.858283367093453 \cdot 10^{-5}$
E2E-SR	2	$(5.0538457150294822 \cdot 10^{-5} - 0.017224455343742888 \cdot x_0) \cdot (-0.0060226605842333864 \cdot x_0 - 58.016864815149059)$
	5	$0.00832 - 0.9720000000000001 \cdot \sin(0.02411009851315413 \cdot x_0 + 0.043594062028465994 - \frac{0.807}{\dots - 9.38 + 0.0027099999999999997})$
	7	$1.0 \cdot \sin(1.00998324807091 \cdot x_0 - 0.00077925815314189615) + 0.0021$
	10	$1.0 \cdot \sin(0.9856914566698666 \cdot x_0 + 0.0016099999999999999 \cdot \arctan(\dots) - 6.2710054014902136) - 0.0$
Operon	2	$1.0 \cdot 1.0 \cdot x_0 + 0.0$
	5	$1.001 \cdot (\cos(-1.0037 \cdot x_0) \cdot 1.2879 \cdot x_0 - \sin(-(\dots - 1.4461 \cdot x_0) - 0.1272 \cdot 0.0001)) + 0.0$
	7	$1.0 \cdot (\cos(-0.4227 \cdot x_0) \cdot 1.2701 \cdot x_0 - (\dots + 0.0 \cdot x_0 - 0.2272) \cdot \sin(1.5034 \cdot x_0)) + 0.0$
	10	$1.001 \cdot (\cos(0.5492 \cdot x_0) \cdot (-0.2811) \cdot x_0 - \sin(-0.9814 \cdot x_0) + \cos(\dots + 0.626 \cdot x_0)) + 0.0$

Table 4.29: Taylor recovery for Feynman I.18.12. Ground truth: $x_0 \cdot x_1 \cdot \sin(x_2)$.

Method	K	Expression
Taylor data	2	$-x_0 \cdot x_1 \cdot x_2 + \pi \cdot x_0 \cdot x_1$
	5	$\dots + \frac{-x_0 \cdot x_1 \cdot x_2^5}{120} + \frac{\pi \cdot x_0 \cdot x_1 \cdot x_2^4}{24} - \frac{\pi^2 \cdot x_0 \cdot x_1 \cdot x_2^3}{12} + \frac{x_0 \cdot x_1 \cdot x_2^3}{6} - \frac{\pi \cdot x_0 \cdot x_1 \cdot x_2^2}{2}$
	7	$\dots + \frac{\pi^3 \cdot x_0 \cdot x_1 \cdot x_2^2}{12} + \frac{x_0 \cdot x_1 \cdot x_2^7}{5040} - \frac{\pi \cdot x_0 \cdot x_1 \cdot x_2^6}{720} - \frac{x_0 \cdot x_1 \cdot x_2^5}{120} + \frac{\pi^2 \cdot x_0 \cdot x_1 \cdot x_2^5}{240} - \frac{\pi^3 \cdot x_0 \cdot x_1 \cdot x_2^4}{144} + \pi \cdot x_0 \cdot x_1 \cdot x_2^4$
	10	$\dots + \frac{-x_0 \cdot x_1 \cdot x_2^9}{362880} + \frac{\pi \cdot x_0 \cdot x_1 \cdot x_2^8}{40320} - \frac{\pi^2 \cdot x_0 \cdot x_1 \cdot x_2^7}{10080} + \frac{x_0 \cdot x_1 \cdot x_2^7}{5040} - \frac{\pi \cdot x_0 \cdot x_1 \cdot x_2^6}{720} + \pi^3 \cdot x_0 \cdot x_1$
MOSAIC-SR	2	$x_0 \cdot x_1 \cdot (\pi - x_2)$
	5	$x_0 \cdot x_1 \cdot \sin(x_2)$
	7	$x_0 \cdot x_1 \cdot \sin(1.002337079977 \cdot x_2)$

Table 4.29: Taylor recovery for Feynman I.18.12. (continued).

Method	K	Expression
	10	$0.564336516744925 \cdot \sqrt{\pi} \cdot x_0 \cdot x_1 \cdot \sin(x_2)$
PySR	2	$x_0 \cdot x_1 \cdot (3.1415925 - x_2)$
	5	$x_0 \cdot (0.33221138 \cdot x_1 \cdot \sin(\cos(x_2)) + x_1) \cdot \sin(\sin(x_2 \cdot 0.4749027 + 1.6496296) \cdot 2.7744522)$ 0.9437794
	7	$x_0 \cdot x_1$ $\cdot \sin(x_2 - -0.019365529 \cdot (x_2 - 3.146233) \cdot \cos(\sin(\sin(\sin(x_2) \cdot (-1.9357585)))) \cdot (-1.7473638)))$
	10	$x_0 \cdot (x_1 \cdot e^{\cos(x_2)} \cdot (-0.0030647963) \cdot \cos(x_2) + x_1) \cdot \sin(x_2 \cdot 0.9981049 - 1 \cdot (-0.0058534783))$
TPSR	2	$(\dots + 0.28331615049921344 + 0.4956534371075346) \cdot (0.34386125723713024 \cdot x_0 - 1.8420104004975268$ $+ 1.8420104004834439)$
	5	$1.276389115335405 + 1$ 1
	7	$\cdot \frac{\dots - 0.07925775134645766 + 0.2495263264800342 + 0.2580259436964331 + 0.2746491984632519}{\sin(-2.215140796977261 \cdot \dots + 0.575852222337376 + 1.7552518942051996 - 0.8214622877077248)}$ $+ 0.610334905734776$
	10	$(\dots + 0.0850328212232805 + 0.1470358948108155) \cdot (0.36255539574241513 \cdot x_0 - 1.8127229580805884$ $+ 1.812829094663855)$
E2E-SR	2	$(-0.002704484223201092 \cdot x_1 - 0.60237161959079474) \cdot (\dots + 0.0011760054997509854 \cdot x_0$ $- 0.0427947554976017 \cdot x_2 + 0.13465458451346508)$
	5	$(0.76120861641426228 \cdot x_0 + 0.022982771994114469) \cdot (0.37355026941984976 \cdot x_1$ $+ 0.17673765134826056) \cdot (\dots - 0.0016428320073234725 \cdot x_0 - 3.12)$
	7	$(169.62829714499832 - 54.02295102376346 \cdot x_2) \cdot (\dots + 1.8694742958078208 \cdot x_1)$ $\cdot (0.001513002143198883 \cdot x_0 + 4.3499570545021802 \cdot 10^{-5})$
	10	$(0.0018476647025342624 \cdot x_0 + 7.4163269681210592 \cdot 10^{-5}) \cdot (-18.721308938254057 \cdot x_1$ $- 3.1062206918614961) \cdot 28.2 \cdot \sin(\dots + 0.9922408707994587)$
Operon	2	$1.0 \cdot 12.6214 \cdot x_0 \cdot (83.0285 - 26.4288 \cdot x_2) \cdot (0.0 \cdot x_0 + 0.003 \cdot x_1) + 0.0$
	5	$1.0 \cdot (-0.9508 \cdot x_2 - -2.9873) \cdot (\dots - 97.4074 \cdot x_0 - 76.7503 \cdot x_0) - 0.0011$
	7	$1.0 \cdot (-0.1636 \cdot x_0 \cdot \sin(-0.539 \cdot x_2 - 1.4483) - 0.7675 \cdot x_2 - (\dots - 2.4113)) - 0.0007$
	10	$1.0 \cdot (0.1933 \cdot x_0 \cdot 5.0729 \cdot x_1 \cdot \sin(1.0105 \cdot x_2) - 0.006 \cdot x_0 \cdot (\dots + 5.3587 \cdot x_1)) + 0.0005$

Table 4.30: Taylor recovery for Feynman I.12.11. Ground truth: $x_0 \cdot (x_1 + x_2 \cdot x_3 \cdot \sin(x_4))$.

Method	K	Expression
Taylor data	2	$x_0 \cdot x_1 - \frac{\sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^2}{4} + \frac{\sqrt{2} \cdot \pi \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4}{8} + \frac{\sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4}{2} - \frac{\sqrt{2} \cdot \pi \cdot x_0 \cdot x_2 \cdot x_3}{8}$ $- \frac{\sqrt{2} \cdot \pi^2 \cdot x_0 \cdot x_2 \cdot x_3}{64} + \frac{\sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3}{2}$
	5	$\dots + x_0 \cdot x_1 + \frac{\sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^5}{240} - \frac{\sqrt{2} \cdot \pi \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^4}{192} + \frac{\sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^4}{48} - \sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3$ $\cdot x_4$
	7	$\dots + x_0 \cdot x_1 - \frac{\sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^7}{10080} - \frac{\sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^6}{1440} + \frac{\sqrt{2} \cdot \pi \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^6}{5760} - \sqrt{2} \cdot \pi^2 \cdot x_0$
	10	$\dots + x_0 \cdot x_1 - \frac{\sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^{10}}{7257600} + \frac{\sqrt{2} \cdot \pi \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^9}{2903040} + \frac{\sqrt{2} \cdot x_0 \cdot x_2 \cdot x_3 \cdot x_4^9}{725760} - \sqrt{2} \cdot \pi \cdot x_0$
MOSAIC-SR	2	$x_0 \cdot (x_1 + 0.712793490396519 \cdot x_2 \cdot x_3 \cdot x_4 + 0.0669510918992685 \cdot x_2 \cdot x_3)$
	5	$x_0 \cdot (x_1 + x_2 \cdot x_3 \cdot \sin(x_4))$
	7	$x_0 \cdot (x_1 + 1.0 \cdot x_2 \cdot x_3 \cdot \sin(x_4))$
	10	$x_0 \cdot (x_1 + x_2 \cdot x_3 \cdot \sin(x_4))$
PySR	2	$x_0 \cdot (x_1 + x_2 \cdot x_3 \cdot (x_4 \cdot (1.262467 + x_4 \cdot (-0.35355335)) - 1 \cdot 0.06634307))$
	5	$x_0 \cdot (x_1 + x_3 \cdot 0.99937403 \cdot x_2 \cdot (0.0004353141 \cdot x_4 + \sin(x_4) + 8.375039 \cdot 10^{-5}))$ $5.59005 \cdot 10^{-8}$
	7	$x_0 \cdot (x_1 + x_2 \cdot x_3 \cdot (\sin(x_4 - \frac{5.59005 \cdot 10^{-8}}{e^{x_4} - 0.98920304}) - \frac{6.089482 \cdot 10^{-8}}{\cos(x_4)}))$

Table 4.30: Taylor recovery for Feynman I.12.11. (continued).

Method	K	Expression
	10	$x_0 \cdot (x_1 + x_2 \cdot x_3 \cdot \sin(x_4))$
TPSR	2	$-(\dots - 0.41703132683721883) + 0.20866626498961413 \cdot (0.3604272691006456 \cdot x_1$ $- 1.744712561451487) - 0.05296867316273429$
	5	$\dots + 0.34294855858724976 \cdot x_1 - 4.723488471036021 - 1.7988515848521762 + 1.7989003805824426$ $+ 2.495998943387973$
	7	$(\dots + 61.51639518156274) \cdot (0.5088734890611518 \cdot (0.17031588558061792 \cdot x_0 - 0.14978961891848902)$ $+ 0.07771255202217539)$
	10	$(\dots + 9.407194864169705) \cdot (0.7224790948921128 \cdot (0.17230290182470367 \cdot x_0 - 0.07264073340143025)$ $+ 0.05248141132993334)$
E2E-SR	2	$(0.1629358158362928 \cdot x_0 - 0.0023126882037319991) \cdot (\dots + 6.16330630162104 \cdot x_1$ $- (-8.0523542289226607 \cdot x_3 - 1.4868480523885724))$
	5	$(-0.31289792906740601 \cdot x_0 - 0.0002607466038638947) \cdot (-3.2888766768517251 \cdot x_1$ $+ 0.3503022089874898 \cdot x_2 - (\dots - 0.06099736784503796 \cdot x_4))$
	7	$(0.093673737069339864 \cdot x_0 - 0.0023842904051689659) \cdot (\dots + 10.452156425909193 \cdot x_1$ $- (2.038660868691718 - 0.5685205786851859 \cdot x_4))$
	10	$(\dots + 0.0014473443753275107 \cdot x_0) \cdot (0.17230290182470367 \cdot x_0 - 0.004740733401430247)$ $+ (0.27324955376110477 \cdot x_4 - 0.97426820283141966)$
Operon	2	$1.0 \cdot (\dots - 2.3002 \cdot x_0 \cdot 0.0 \cdot x_3 + 0.0102 \cdot x_0 \cdot (97.7415 \cdot x_1 + 0.0 \cdot x_2)) + 0.0$
	5	$1.0 \cdot (2.3026 \cdot x_1 \cdot 0.4342 \cdot x_0 + 0.0133 \cdot x_0 \cdot 23.3392 \cdot x_4 \cdot (\dots + 0.0236 \cdot x_2)) - 0.0025$
	7	$1.0021 \cdot (4.6741 \cdot x_0 - (0.6931 \cdot x_4 \cdot (-0.0536) \cdot x_4 \cdot 102.1899 \cdot x_3 - (\dots - 29.3384 \cdot x_3))) - 0.4812$
	10	$1.0 \cdot (-0.0658) \cdot x_4 \cdot 1.1554 \cdot x_3 \cdot (\dots - 0.1744 \cdot x_0 \cdot (-5.8297) \cdot x_4) - 0.0127$

4.1.7 Validation NMSE Trajectories during Local Search

Figures 4.6–4.11 track the best validation NMSE achieved so far throughout each MOSAIC-SR local search run. Lower values are better and the vertical axis is logarithmic. Thin gray step curves show individual runs, the colored curve is the median, and the shaded region is the interquartile range. The dashed horizontal line marks $\text{NMSE} = 10^{-3}$, equivalent to $R^2 > 0.999$, which is the accuracy threshold used for SRSD-Feynman. To distinguish target difficulty from proposal quality, each split is analyzed first by ground truth operator count and then by the operator count of the top decoded sketch. Operator count is used as a proxy for structural size. It is positively related to expression length and complexity but is not identical to either. Easy and Medium trajectories are shown for 50 seconds, while Hard trajectories are shown for 100 seconds. All three splits use five seeds.

SRSD-Feynman Easy

Ground Truth Operator Count. Figure 4.6 first groups runs by the number of operators in the true equation. Equations with fewer operators reach low NMSE earlier. The three and four operator groups nevertheless cross the accuracy threshold soon afterward and converge to the same low error regime.

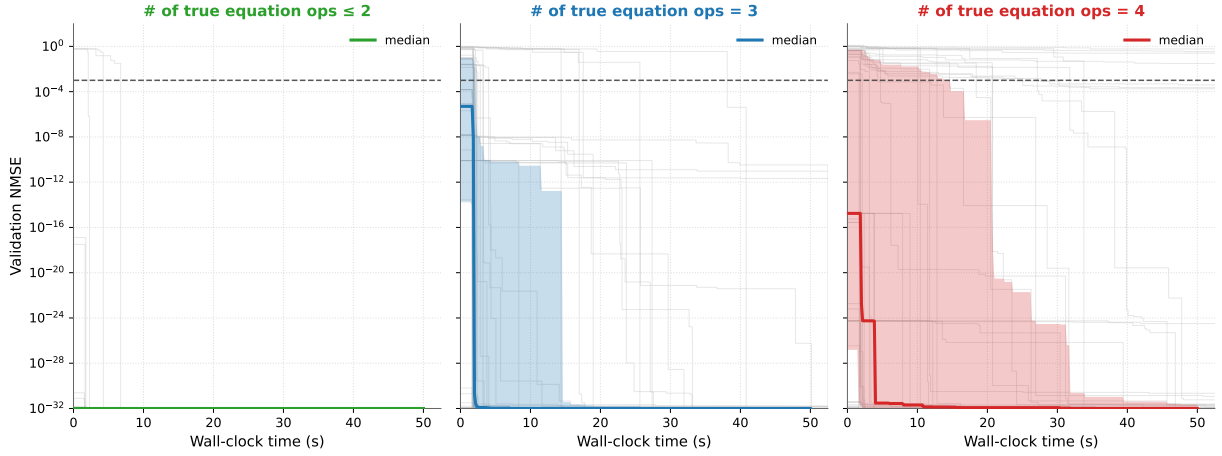


Figure 4.6: Best validation NMSE versus wall clock time on SRSD-Feynman Easy, grouped by ground truth operator count. The 150 runs comprise 30 equations evaluated over five seeds. The groups contain one or two, three, and four true operators, with 40, 55, and 55.

Decoded sketch operator count. Figure 4.7 shows a much stronger separation by decoded sketch operator count. Sketches with fewer than 10 operators cross the threshold within a few seconds, while those with 10-15 operators converge more slowly and some runs with more than 15 operators remain above it. Because Easy targets contain at most four operators, high decoded operator counts appear to constrain local search more than ground truth operator count.

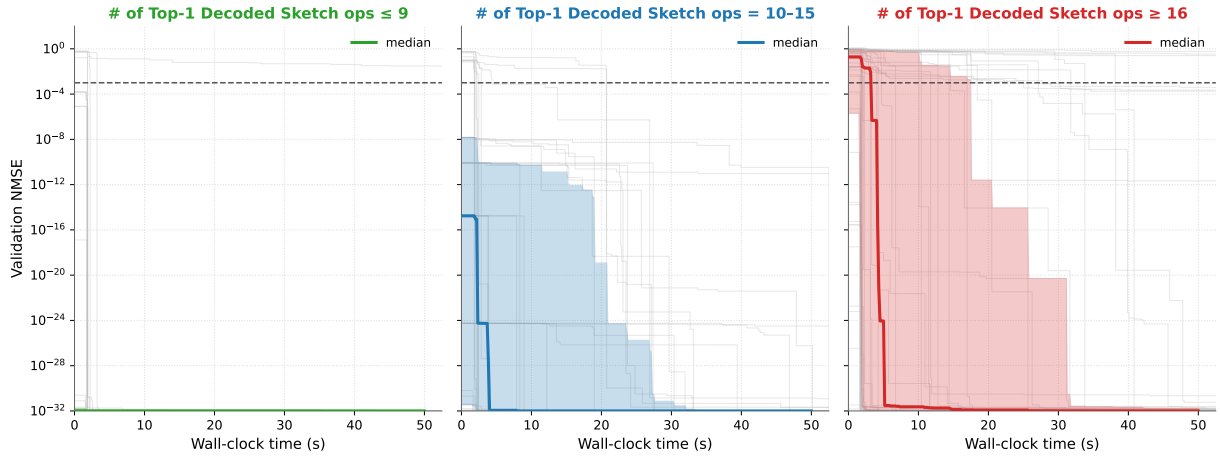


Figure 4.7: Best validation NMSE versus wall clock time on SRSD-Feynman Easy, grouped by top decoded sketch operator count. The runs are divided into sketches with fewer than 10, between 10 and 15, and more than 15 operators, with 57, 44, and 49 runs respectively.

SRSD-Feynman Medium

Figure 4.8 groups runs by the number of operators in the true equation. Equations with fewer than five operators reach low NMSE earlier, while the two higher count groups improve more slowly and show wider variation. Their medians are not strictly ordered by operator count, indicating that operator count captures only one component of target difficulty. The remaining variation depends on the specific algebraic structure of each equation.

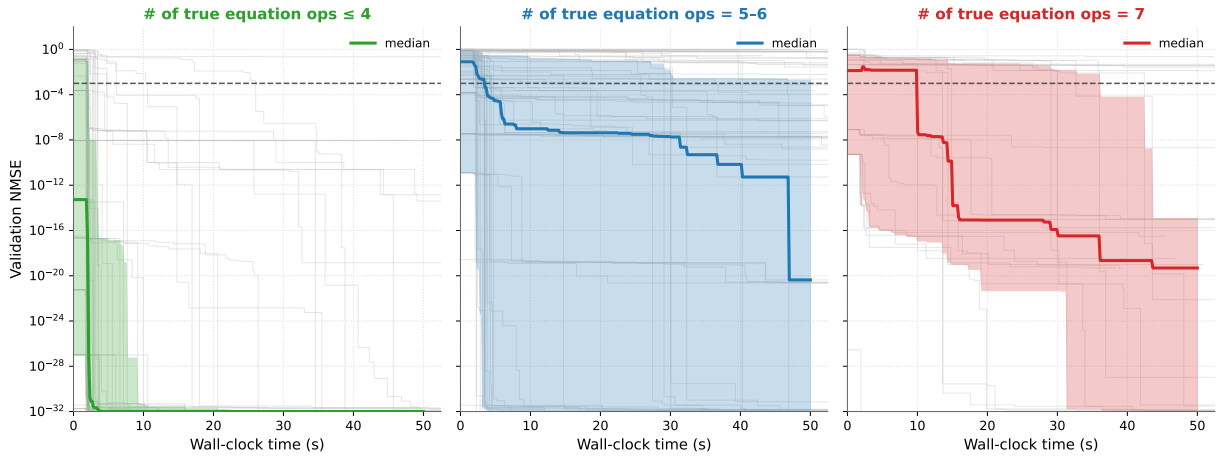


Figure 4.8: Best validation NMSE versus wall clock time on SRSD-Feynman Medium, grouped by ground truth operator count. The 200 runs comprise 40 equations evaluated over five seeds. The groups contain fewer than five, five or six, and more than six true operators, with 75, 95, and 30 runs, respectively.

Decoded sketch operator count. Figure 4.9 shows a clearer separation by decoded sketch operator count. Sketches with fewer than 13 operators cross the accuracy threshold in about 10 seconds, those with 13–17 operators require closer to 30 seconds, and the median for sketches with more than 17 operators remains above it. Decoded sketch operator count therefore provides a stronger indication of local search progress.

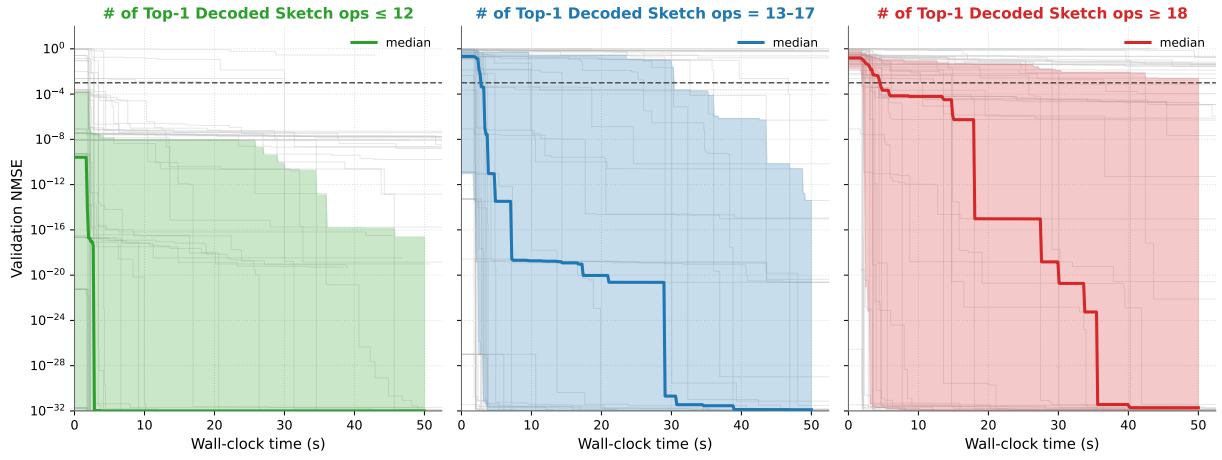


Figure 4.9: Best validation NMSE versus wall clock time on SRSD-Feynman Medium, grouped by top decoded sketch operator count. The groups contain fewer than 13, between 13 and 17, and more than 17 decoded operators, with 82, 53, and 65 runs respectively.

SRSD-Feynman Hard

Ground Truth Operator Count. Figure 4.10 groups runs by the number of operators in the true equation. The medians for equations with fewer than nine operators and those with nine to 11 operators both cross the accuracy threshold early. The lower count group subsequently reaches much lower NMSE. The median for equations with more than 11 operators improves gradually but remains above the threshold after 100 seconds. Ground truth operator count therefore captures an important component of local search difficulty on Hard.

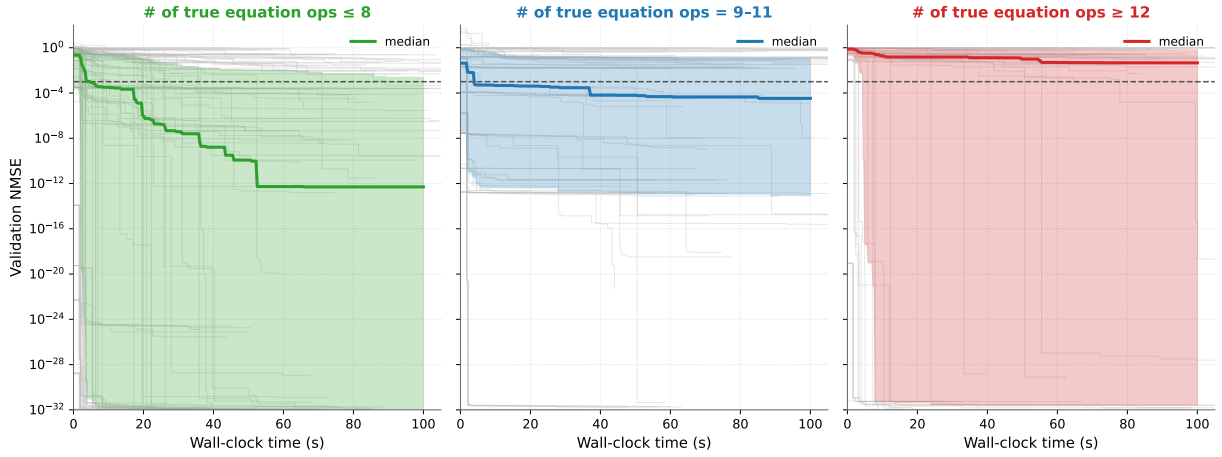


Figure 4.10: Best validation NMSE versus wall clock time during the 100 second search window on SRSD-Feynman Hard, grouped by ground truth operator count. The 250 runs comprise 50 equations evaluated over five seeds. The groups contain fewer than nine, between nine and 11, and more than 11 true operators, with 115, 75, and 60 runs, respectively.

Decoded sketch operator count. Figure 4.11 shows a stronger separation by decoded sketch operator count. The median for sketches with fewer than 13 operators crosses the accuracy threshold almost immediately. The median for sketches with 13-19 operators crosses after about 40 seconds. The median for sketches with more than 19 operators remains above the threshold until the end of the 100 second window. A high decoded operator count therefore slows local search and may require the full available budget to reach the threshold.

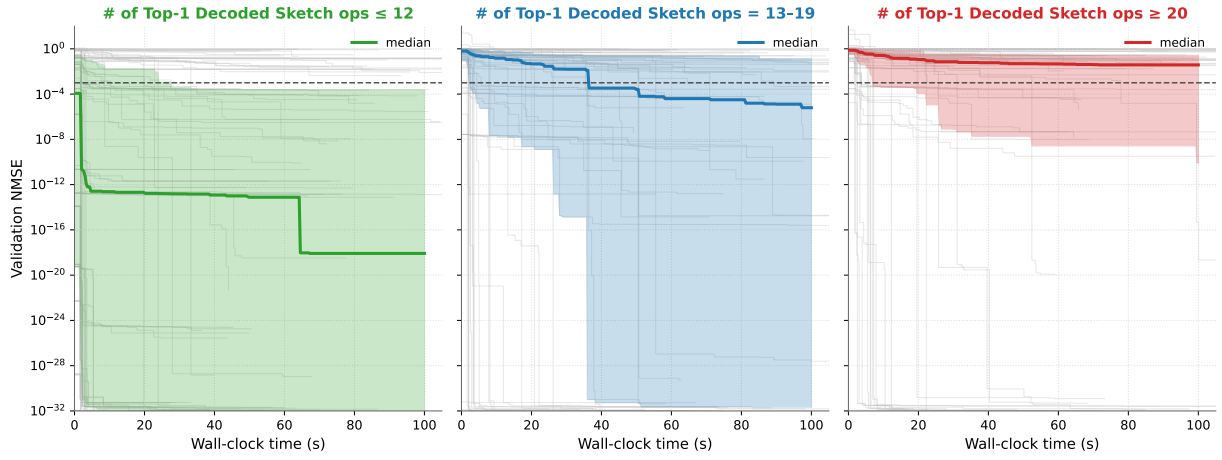


Figure 4.11: Best validation NMSE versus wall clock time on SRSD-Feynman Hard, grouped by top decoded sketch operator count. The x axis spans 0–100 seconds. The 250 runs comprise 50 equations evaluated over five seeds. The groups contain fewer than 13, between 13 and 19, and more than 19 decoded operators, with 107, 73, and 70 runs, respectively.

Overall implication. Across all three splits, higher ground truth operator counts generally slow convergence. In comparison, decoded sketch operator count produces a stronger and more consistent separation of local search outcomes. This finding is practically useful. Ground truth operator count is unknown at inference time. However, decoded sketch operator count is directly observable. These results motivate operator count aware decoding and resampling of sketches with unusually high operator counts. They also support adaptive allocation of the local search budget based on proposal size.

4.1.8 Ablation Studies

We conduct the ablations on the Easy split of SRSD-Feynman, which contains 30 equations. Each result reports symbolic recovery under the same verification protocol as the main SRSD-Feynman experiment, together with numerical recovery. The component ablation is evaluated with three method seeds: 0, 1, and 2.

Initialization and search. Table 4.31 separates the roles of initialization and search. Decoder disables MCTS and local search. Full MOSAIC adds 24 search paths with 300 mutations each after MCTS. Random + search replaces learned sketches with random sketches and uses comparable candidate counts. MCTS, constant fitting, local search, and search budgets are unchanged. Across three seeds, Full MOSAIC reaches 97.8% Acc and 85.6% Sym, versus 31.1% and 15.6% for Random + search. Learned sketches with MCTS but no local search reach only 30.0–43.3% Sym. Both learned initialization and subsequent repair improve recovery on this split.

Table 4.31: Component ablation on 30 clean SRSD-Feynman Easy equations. Acc and Sym report numerical and symbolic recovery in percent. All variants use seeds 0, 1, and 2.

Seed	Decoder		Decoder + MCTS		Full MOSAIC		Random + search	
	Acc	Sym	Acc	Sym	Acc	Sym	Acc	Sym
0	60.0	36.7	66.7	43.3	96.7	83.3	36.7	20.0
1	46.7	30.0	60.0	30.0	96.7	83.3	26.7	10.0
2	46.7	30.0	66.7	40.0	100.0	90.0	30.0	16.7

Search breadth and depth. Table 4.32 varies the number of independent search paths and the number of mutation steps per path. The reported recovery rates are averaged over three method seeds. Recovery generally improves with both search breadth and trajectory depth. At 500 steps, increasing the number of paths from one to 48 raises numerical

recovery from 80.0% to 98.9% and symbolic recovery from 56.7% to 87.8%. The gains from additional steps become smaller at larger search budgets, with numerical recovery at 48 paths reaching 98.9% after 300 steps and remaining unchanged at 500 steps, while symbolic recovery increases from 85.6% to 87.8%. The gains from increasing the number of paths from 24 to 48 are also modest, improving numerical recovery by 1.1 percentage points and symbolic recovery by 2.2 percentage points at 500 steps.

Table 4.32: Search budget ablation on 30 SRSD-Feynman Easy equations. Acc and Sym report numerical and symbolic recovery. All values are percentages. Paths are independent, and the columns are snapshots from continuous trajectories.

Paths	100 steps		300 steps		500 steps	
	Acc	Sym	Acc	Sym	Acc	Sym
1	70.0	47.8	76.7	53.3	80.0	56.7
6	77.8	57.8	87.8	70.0	90.0	75.6
12	83.3	66.7	93.3	76.7	94.4	82.2
24	84.4	70.0	97.8	85.6	97.8	85.6
48	87.8	73.3	98.9	85.6	98.9	87.8

4.1.9 Discussion

Section 4.1 developed MOSAIC-SR as a hybrid approach to exact symbolic recovery. A pretrained Transformer identifies promising expression structures instead of relying on random initialization. Conditional MCTS completes partially specified sketches, and parallel local searches alternate structural repair with constant refitting. This repeated interaction corrects errors in both symbolic structure and numerical scale.

Across the reported benchmarks, MOSAIC-SR achieves the highest symbolic solution rate while retaining competitive predictive accuracy. Its advantage persists when irrelevant variables are introduced. This supports the use of learned sketches as search priors rather than direct predictions. The runtime results further suggest that a learned sketch prior can reduce search time while supporting higher symbolic solution rates. The experiments and diagnostic analyses therefore show that the neural model and explicit search have complementary roles, without treating the decoder as the final decision rule.

Recovery nevertheless becomes more difficult for high dimensional equations because the corresponding search space grows rapidly. Targets far outside the pretraining distribution may also require a larger search budget. Natural extensions include broader pretraining distributions and recovery from sparse observations, which more closely reflect data availability in physical systems.

Section 4.2 considers a different source of information for symbolic model selection: stability across environments.

4.2 Residual-Invariant Selection of Symbolic Models

4.2.1 Motivation and Scope

This section presents RIMS as a finite candidate selector for symbolic regression across environments, together with its analysis, identification boundary, and empirical evaluation.

Section 4.1 addressed the search side of symbolic regression: it used a learned prior and explicit repair to increase the chance that a correct expression is generated. This section addresses a different failure. Even when the correct mechanism is already present in the candidate pool, pooled numerical fit can select the wrong expression.

This problem can arise naturally in scientific data collection. The direct causes of a response are rarely known before the experiment, so investigators record a broad panel of candidate covariates, such as sensor readings, biomarkers, diagnostics, and intermediate quantities. Some recorded variables may be causal parents of the response, while others may have been generated downstream of it. A symbolic regression system that is given all recorded covariates can then use a downstream measurement as a shortcut instead of recovering the mechanism from its parents.

Consider a stable target mechanism

$$Y = f^*(X_1, X_2) + \varepsilon \quad (4.10)$$

and an observed downstream measurement

$$D^e = \rho_e Y + \eta^e, \quad (4.11)$$

where e indexes the data collection environment, ρ_e is the measurement coupling, and η^e is measurement noise. An equation using D^e can predict Y accurately because D^e contains a noisy copy of the target, and it may be simpler than f^* . Such an equation is a *descendant shortcut*: it predicts the target through a variable measured downstream of the target rather than through its generating parents in the structural causal model sense [114, 118].

The signal used below is a change in the shortcut’s *residual law*, not necessarily a change in the structural channel $P(D \mid Y, e)$ itself. The residual law can change because the measurement calibration ρ_e changes, or because a stable noisy channel is combined with a shift in $P(Y \mid e)$ and the candidate expression is only an imperfect inverse of that channel. In either case, a shortcut can have environment-dependent residuals while the stable mechanism leaves only the environment-invariant noise ε .

Our method, Residual Invariant Model Selection (RIMS), uses this distinction. It is a post hoc selector for a finite symbolic *hall of fame*. It does not generate expressions, refit their constants, or replace the symbolic regression backend. It retains candidates with nearly optimal pooled fit and then chooses the candidate whose residual distribution is most stable across environments. This idea is related to invariant causal prediction [117], but the selected object is different. Invariant causal prediction tests variable sets; RIMS ranks fully specified symbolic equations. The residual distribution criterion is especially close to nonlinear invariant prediction [53].

RIMS can help when environment labels are available, the target mechanism and its noise law are stable, the symbolic regression search has already produced a fit-competitive mechanism, and competing shortcuts create a detectable cross-environment residual-law gap. It is a finite-candidate selector, not a general causal discovery algorithm, and it does not guarantee improved out of distribution prediction.

4.2.2 Problem Setting

Suppose observations are collected in E *environments*. An environment is a data collection regime, such as an experimental intervention, instrument, batch, site, time period, or operating condition. The index e identifies the regime and is not supplied as an input to the symbolic expressions. Environment e supplies

$$\mathcal{D}^e = \{(x_i^e, y_i^e)\}_{i=1}^{n_e}, \quad e = 1, \dots, E. \quad (4.12)$$

Across the environments used for selection, the target obeys the stable structural equation

$$Y^e = f^*(X_{S^*}^e) + \varepsilon^e, \quad \varepsilon^e \sim P_\varepsilon, \quad \varepsilon^e \perp\!\!\!\perp X_{S^*}^e. \quad (4.13)$$

Here X^e contains all observed inputs and S^* indexes the physical parents. The noise is independent of these parents within each environment. Descendant inputs may share ε^e . The parent distribution may change with e . The mechanism f^* and noise law P_ε stay fixed. Randomly assigning environment labels is generally not useful: the labels must correspond to a distribution change that can expose the shortcut while preserving the target mechanism.

Let

$$\mathcal{H} = \{g_1, \dots, g_K\} \quad (4.14)$$

be the frozen hall of fame returned by a symbolic regression backend such as PySR or Operon [29, 73]. Each g_k includes its expression structure and fitted constants. The environment label is not an input variable available to these expressions.

Formally, an expression $g_D(D)$ is an environment-dependent descendant shortcut when D is downstream of Y , g_D is competitive in pooled fit, and the residual law

$$\mathcal{L}(Y - g_D(D) \mid e) \quad (4.15)$$

varies with the environment. This definition is stated in terms of the induced residual law; the downstream measurement channel itself need not change.

This setting separates *search failure* from *selection failure*. Search fails when f^* , or an equivalent expression, is absent from $\mathcal{H}$. No reranking rule can recover a candidate that was never generated. Selection fails when the mechanism is present but the backend's pooled fit or fit-complexity criterion chooses a shortcut rather than the correct causal expression. RIMS targets only the second failure.

4.2.3 Residual Invariant Model Selection

For a candidate $g \in \mathcal{H}$, define its residual in environment e by

$$r_i^e(g) = y_i^e - g(x_i^e). \quad (4.16)$$

The empirical pooled risk is

$$\hat{R}(g) = \frac{1}{\sum_e n_e} \sum_{e=1}^E \sum_{i=1}^{n_e} r_i^e(g)^2. \quad (4.17)$$

Pooled risk remains important: an invariant but numerically poor expression should not be selected. RIMS therefore treats fit as an admissibility condition rather than combining fit and invariance with a tuned scalar penalty.

For a fixed candidate g , let $\hat{F}_e^g$ be the empirical cumulative distribution function of its residuals in environment e . For a pair of environments $e < e'$, let $S_{ee'}^g$ be their pooled residual sample. The normalized two-sample Cramér–von Mises discrepancy is

$$\hat{d}_{ee'}(g) = \frac{1}{n_e + n_{e'}} \sum_{z \in S_{ee'}^g} \left(\hat{F}_e^g(z) - \hat{F}_{e'}^g(z) \right)^2. \quad (4.18)$$

This normalized form lies in $[0, 1]$. It is dimensionless, requires no bounded residual support, and depends on the entire residual distribution rather than only its mean or variance. The empirical *residual invariance score* averages over environment pairs,

$$\hat{I}(g) = \frac{1}{\binom{E}{2}} \sum_{1 \leq e < e' \leq E} \hat{d}_{ee'}(g). \quad (4.19)$$

Uniform pair weights are used by default. Other two-sample discrepancies can replace Equation (4.18). The method relies on residual distribution invariance rather than on CvM specifically.

Let $\hat{R}_{\min} = \min_{h \in \mathcal{H}} \hat{R}(h)$. For a relative fit tolerance $\epsilon \geq 1$ and an optional absolute floor $\tau_R \geq 0$, define the *fit-competitive set*

$$\hat{\mathcal{H}}_{\epsilon, \tau_R} = \left\{ g \in \mathcal{H} : \hat{R}(g) \leq \epsilon \hat{R}_{\min} + \tau_R \right\}. \quad (4.20)$$

The absolute floor prevents a purely relative window from becoming vacuous in near-noiseless problems. The experiment uses $\tau_R = 0$. Plain RIMS returns

$$\hat{g}_{\text{RIMS}} \in \arg \min_{g \in \hat{\mathcal{H}}_{\epsilon, \tau_R}} \hat{I}(g). \quad (4.21)$$

The general default is $\epsilon = 1.1$. Thus a candidate must be within ten percent of the best empirical fit before invariance can change the ranking. The Feynman test below also examines wider tolerances because the planted sensor has a systematic fit advantage.

Algorithm 1 summarizes the proposed selector. To obtain the finite-candidate interpretation used in the theory, candidate search and RIMS scoring should use disjoint data. Candidate structures and constants are frozen after search; otherwise the scoring data would also influence the candidate pool.

Algorithm 1 Plain RIMS.

Require: Hall of fame $\mathcal{H}$ frozen on a search split; environment-labeled scoring split; fit tolerances $\epsilon \geq 1$ and $\tau_R \geq 0$

- 1: Keep the structure and constants of every $g_k \in \mathcal{H}$ fixed; do not refit candidates on the scoring split.
 - 2: Compute $\hat{R}(g_k)$ and $\hat{I}(g_k)$ using Equations (4.17) and (4.19).
 - 3: Form $\hat{\mathcal{H}}_{\epsilon, \tau_R} = \{g_k : \hat{R}(g_k) \leq \epsilon \hat{R}_{\min} + \tau_R\}$.
 - 4: **return** $\arg \min_{g \in \hat{\mathcal{H}}_{\epsilon, \tau_R}} \hat{I}(g)$.
-

Two optional extensions are useful in some applications but are not part of plain RIMS. RIMS-C uses expression complexity as a deterministic tie break inside a small near-invariance band. RIMS-Safe abstains when the leading invariance scores are insufficiently separated. These extensions do not change the selector above and are not used for the results reported here.

RIMS adds

$$\mathcal{O}(KE^2\bar{n}\log\bar{n}) \quad (4.22)$$

work for K candidates, E environments, and typical environment size $\bar{n}$. The sorting needed to form empirical CDFs gives the logarithmic factor. In a representative setting with ten environments and 512 observations per environment, the median reranking time is 0.48 seconds and the 95th percentile is 0.65 seconds. The corresponding PySR search uses a budget of up to 120 seconds per task, so the post hoc selection cost is small relative to candidate generation.

4.2.4 Finite-Candidate Guarantee and Identification Boundary

The theory treats RIMS as a selector over a fixed finite pool. Let $\bar{R}(g)$ and $I(g)$ be the population counterparts of Equations (4.17) and (4.19), and let $R_* = \min_{g \in \mathcal{H}} \bar{R}(g)$. Suppose that, on an event Ω of high probability,

$$\sup_{g \in \mathcal{H}} |\hat{R}(g) - \bar{R}(g)| \leq \alpha_R, \quad \sup_{g \in \mathcal{H}} |\hat{I}(g) - I(g)| \leq \alpha_I. \quad (4.23)$$

For a pool fixed independently of the scoring split, the bounded normalized CvM score admits distribution-free concentration through empirical CDF bounds, with

$$\alpha_I = \mathcal{O}\left(\sqrt{\frac{\log(KE/\delta)}{n_{\min}}}\right). \quad (4.24)$$

Concentration of squared-error risk additionally requires a bounded or clipped loss, or a suitable light-tail condition on squared residuals. A sample split or cross-fitting makes $\mathcal{H}$ fixed conditional on the search data and therefore gives the setting covered by this finite-pool argument.

Because algebraically equivalent expressions can represent the same mechanism, the recovery statement is formulated for a *mechanism class*. Let $\mathcal{M} \subseteq \mathcal{H}$ denote all candidates counted as correct under the chosen equivalence or Match@ τ criterion. Define the population risk envelope

$$\mathcal{B}^+ = \{g \in \mathcal{H} : \bar{R}(g) \leq \epsilon R_* + \tau_R + (1 + \epsilon)\alpha_R\}. \quad (4.25)$$

Theorem 4.1 (Mechanism-class recovery). *Assume Equation (4.23). Suppose that some $m^\circ \in \mathcal{M}$ is safely fit admissible,*

$$\bar{R}(m^\circ) \leq \epsilon R_* + \tau_R - (1 + \epsilon)\alpha_R, \quad (4.26)$$

and that every admissible non-mechanism $g \in \mathcal{B}^+ \setminus \mathcal{M}$ satisfies

$$I(g) \geq I(m^\circ) + \gamma \quad \text{for some } \gamma > 2\alpha_I. \quad (4.27)$$

Then plain RIMS selects a member of $\mathcal{M}$ on Ω .

Proof. Uniform risk concentration and Equation (4.26) place m° in the empirical fit set, while every member of the empirical fit set lies in $\mathcal{B}^+$. Hence the smallest empirical invariance score in the fit set is at most $I(m^\circ) + \alpha_I$. For every $g \in \mathcal{B}^+ \setminus \mathcal{M}$, Equation (4.27) gives $\widehat{I}(g) \geq I(m^\circ) + \gamma - \alpha_I > I(m^\circ) + \alpha_I$. No non-mechanism can therefore attain the empirical minimum, so the selected candidate belongs to $\mathcal{M}$. $\square$

An exact mechanism candidate has residual ε^e , with the same marginal law P_ε in every environment. Its population invariance score is zero. Theorem 4.1 uses only the uniform error bounds, safe fit condition, and residual gap. It does not require noise independence from descendant inputs. Shared noise may give a shortcut lower pooled risk. The margin in Equation (4.26) accounts for estimation error. The guarantee still requires an available mechanism inside the fit window and a detectable residual gap.

Residual invariance also has a limitation.

Proposition 4.2 (Invariant downstream channels are not identifiable from residual invariance). *Suppose a fit-competitive candidate g_D using a downstream variable has the same residual law in every environment. Then $I(g_D) = 0$. If the true mechanism also has an invariant residual law, residual invariance alone cannot prefer the mechanism to g_D .*

Proof. Every pairwise distribution discrepancy in Equation (4.19) is zero for g_D . Hence its score is the same as that of any other candidate with an invariant residual law. $\square$

For example, if $D^e = \rho Y^e + \eta^e$ with fixed $\rho \neq 0$ and an environment-invariant noise law for η^e , then the exact inverse $g_D(D) = D/\rho$ has residual $Y^e - g_D(D^e) = -\eta^e/\rho$, which is invariant. No selector based only on the residual invariance score can distinguish this stable inverse from the target mechanism. The proposition is deliberately stated in terms of the residual law: constant channel parameters alone do not guarantee non-identifiability. For example, applying h^{-1} to a nonlinear post-link noisy measurement $D = h(Y) + \eta$ need not produce invariant residuals when $P(Y | e)$ changes. Side information, a justified complexity prior, or abstention is needed only when a fit-competitive downstream candidate actually has an invariant residual law.

The theorem applies directly when candidate search and RIMS scoring use disjoint data, as described above. The Feynman quantile bin study is not a direct application of the concentration result because its environments are constructed from the realized data.

4.2.5 Experiments

The empirical evaluation focuses on the official AI Feynman datasets. It tests whether residual invariance can recover physical mechanisms when a planted measurement shortcut dominates pooled fit.

Feynman Equations Test

Design. The experiment uses the official data points and target equations of all 119 PMLB AI Feynman datasets, comprising 99 main equations and 20 additional benchmark equations [154]. The physical covariates and noiseless targets are retained. Training labels receive noise. A measurement shortcut and one irrelevant column are added.

For each equation, rows are sorted by the first physical parent x_0 and split into ten equal-count quantile bins. A bin is therefore a regime with a distinct range of x_0 , not a random label. Bins $\{0, 1, 2, 3, 5, 6, 7, 8\}$ are used for search and selection, bin 4 is a held-out interior regime, and bin 9 is a held-out extreme regime. Each training bin is subsampled to 512 points. The mechanism is unchanged across bins, and the partition is never defined from the target. Evaluation is Match@5% against the noiseless official target on bins 4 and 9.

Here $Y = f^*(X_{S^*})$ is the noiseless official target. The training response corresponding to Y^e in Equation (4.13) is

$$\tilde{Y} = Y + \gamma \text{RMS}(Y)\xi, \quad \gamma = 0.1, \quad \xi \sim \mathcal{N}(0, 1). \quad (4.28)$$

For the fixed benchmark data, $\gamma \text{RMS}(Y)$ is constant across bins and ξ is independent of the physical inputs. The *leakage sensor* reads this same noisy response:

$$D^e = g_e \tilde{Y} + \sigma_{\text{abs}} \eta, \quad \sigma_{\text{abs}} = c \bar{g} \gamma \text{RMS}(Y), \quad c = 0.9, \quad (4.29)$$

Here $\eta \sim \mathcal{N}(0, 1)$ is independent of ξ and the physical inputs. Thus $\varepsilon^e = \gamma \text{RMS}(Y)\xi$ is independent of the parents but dependent on D^e , an observed input. For each equation, $s \sim \text{Uniform}(0.6, 0.95)$, $g_e = sq_e$, and $\bar{g} = E^{-1} \sum_e g_e$. The vector $(q_0, \dots, q_9)$ is a random permutation of ten equally spaced values in $[0.9, 1.0]$. This avoids a monotone gain trend in x_0 . Bins 4 and 9 use gains unseen during training. An expression using D^e can exploit the shared noise to fit $\tilde{Y}$ better than the physical mechanism and enter the PySR hall of fame.

This fit advantage also explains why the Feynman study uses a wider fit window. For the competitive sensor noise level $c = 0.9$, the analytic truth-to-best-hybrid fit ratio is

approximately $(1 + c^2)/c^2 = 2.23$. The diagnostic value $\epsilon = 2.5$ was therefore fixed before inspecting the union-pool recovery outcomes; the general default remains $\epsilon = 1.1$.

The leakage sensor can poison search by displacing the mechanism from the hall of fame. To restore candidate availability, the symbolic regression backend is run $p + 1$ times for a problem with p observed columns: once using all columns and once after dropping each column in turn. The resulting halls of fame are merged into a leave-one-feature-out (LOFO) union pool. No column is identified as the sensor in advance. A run that happens to omit the sensor can rediscover the physical mechanism, while the full-feature run supplies the shortcut candidates; RIMS then performs selection from their union.

Results. Table 4.33 reports both recovery and the number of selected expressions that contain the planted sensor variable.

Selector	Recovery	Selected expressions using sensor
Pooled fit	0/119	119/119
RIMS, $\epsilon = 1.1$	2/119	117/119
RIMS, $\epsilon = 1.5$	15/119	99/119
RIMS, $\epsilon = 2.0$	50/119	51/119
RIMS, $\epsilon = 2.5$	73/119	20/119
RIMS, $\epsilon = 3.0$	73/119	20/119
Mechanism available in union pool	81/119	–

Table 4.33: Recovery and planted sensor use on the 119 official Feynman datasets. Recovery is Match@5% on the test interior and extreme bins.

Pooled selection uses the sensor and fails on all 119 equations. The LOFO union contains a matching physical mechanism for 81 equations. At $\epsilon = 2.5$, RIMS recovers 73 of these 81 mechanisms and incurs no loss relative to pooled selection, giving an exact one-sided sign-test value of 1.1×10^{-22} . The mechanism is absent from the union pool for 38 equations, so no post hoc selector can recover it there. Eight additional equations with an available mechanism are not selected by RIMS at this tolerance. This experiment therefore separates the two parts of the failure: LOFO restores candidate availability, while residual invariance resolves most of the remaining selection problem.

4.2.6 Discussion

RIMS is appropriate when the data provide meaningful environments, the target mechanism is stable, and a symbolic backend returns several fit-competitive candidates. Its

main use is to reject a downstream or leakage-based expression that induces a detectable cross-environment residual-law gap.

The method should not be interpreted as a generic causal certificate. Its success is gated by three conditions. First, the mechanism must be available in the hall of fame. Second, it must remain inside the fit-competitive set. Third, every competing shortcut must be separated from the mechanism by a detectable residual-invariance gap. These correspond to search failure, fit-admissibility failure, and non-identifiability, respectively. A descendant that admits a fit-competitive invariant inverse cannot be distinguished by residual invariance alone.

Environment quality is equally important. An *arbitrary* environment label is a partition unrelated to the relevant data-generating change, such as a random split that leaves the shortcut relationship essentially unchanged. Such labels provide no systematic invariance signal. RIMS is also unreliable when shifts are too weak, when residual noise overwhelms the separation, or when the target mechanism or its noise law changes across environments. Lower residual instability by itself is not evidence of causal recovery or improved out-of-distribution prediction.

Future work will evaluate RIMS on a broader range of datasets and on candidate pools produced by additional symbolic regression backends, under varied environmental shifts. Another direction is to incorporate residual invariance into symbolic search itself rather than using it only to rerank a completed candidate pool. These directions are discussed further in Chapter 5.

Sections 4.1 and 4.2 are therefore complementary. MOSAIC-SR attempts to improve candidate generation and repair, which addresses mechanism availability. RIMS assumes that search has already produced a useful finite pool and improves the final decision using information across environments. Together they separate two questions that pooled symbolic regression often conflates: whether the correct equation can be found, and whether it can be recognized after it has been found.

Chapter 5

Conclusion

5.1 Summary of Contributions

This thesis studied how structure can make Transformer computation and scientific model discovery more efficient and reliable. Fast Multipole Attention (FMA) uses geometric structure to organize information exchange, MOSAIC-SR uses symbolic structure to guide equation search, and Residual Invariant Model Selection (RIMS) uses environmental structure to select among candidate equations.

Chapter 3 introduced FMA, a multilevel attention operator for sequences and images. Exact local interactions are retained in a near field, while distant interactions are represented by learned summaries on an interval hierarchy or quadtree. This gives every layer a global receptive field with loglinear or linear rather than quadratic complexity. FMA achieved the best accuracy among the evaluated efficient attention baselines on enwik8 and WikiText-103 and remained close to full attention at long context lengths. In vision, FMA achieved higher accuracy while using less memory than comparable dense attention baselines on image classification and semantic segmentation.

MOSAIC-SR was introduced in Section 4.1. Its pretrained Transformer proposes equation sketches, conditional MCTS completes them, and parallel local searches alternate structural repair with constant refitting. Treating neural generation as a search prior rather than a final prediction improved exact recovery while retaining competitive predictive accuracy, including with irrelevant variables. MOSAIC-SR obtained the highest symbolic solution rate on every reported benchmark.

Section 4.2 introduced RIMS for the case in which search has produced the correct mechanism but pooled fit selects a shortcut. RIMS reranks fit-competitive candidates

by residual distribution invariance across environments. On the 119 official AI Feynman datasets with a planted measurement sensor, pooled fit recovered no physical mechanisms, whereas RIMS recovered 73 mechanisms at $\epsilon = 2.5$. The physical mechanism was available in the union candidate pool for 81 datasets. Thus, MOSAIC-SR and RIMS address complementary questions: whether the correct equation can be found, and whether it can be recognized after it is found.

5.2 Limitations and Open Problems

FMA is not exact dense attention and depends on a meaningful sequence or spatial hierarchy. For short inputs, or data without useful geometry, optimized dense attention may be preferable. Also, a general approximation theory for learned attention remains open.

MOSAIC-SR becomes more difficult to apply to high dimensional equations because the corresponding search space grows rapidly. Targets far outside the pretraining distribution may also require a larger search budget. Because RIMS is a post hoc selector, it cannot recover a mechanism that is absent from the candidate pool.

5.3 Future Directions

For FMA, an important engineering direction is to integrate IO aware tiling and fused kernels into both the near field and far field computations. This could translate the lower operation count and memory use into larger wall clock improvements. Combining the hierarchy with rotary or relative positional encodings may also improve long range extrapolation in language models. The same construction can be extended from quadtrees to octrees for video and volumetric images.

For MOSAIC-SR, the main priority is to broaden the pretraining distribution so that the learned sketch prior covers higher dimensional equations with more complex symbolic structure. A second direction is recovery from sparse observations, which better reflects data availability in real world physical systems. These extensions would test whether the learned prior can continue to focus search when the expression space is larger and the numerical evidence is limited.

For RIMS, future work will evaluate the selector on a broader range of datasets and on candidate pools produced by additional symbolic regression backends, under varied environmental shifts. A second direction is to incorporate residual invariance into symbolic search rather than applying it only after a candidate pool has been produced.

References

- [1] Joshua Ainslie et al. ETC: Encoding long and structured inputs in transformers. In *Conference on Empirical Methods in Natural Language Processing*, 2020.
- [2] Jean-Baptiste Alayrac et al. Flamingo: a visual language model for few-shot learning. In *Advances in Neural Information Processing Systems*, 2022.
- [3] T. W. Anderson and D. A. Darling. A test of goodness of fit. *Journal of the American Statistical Association*, 1954.
- [4] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.
- [5] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A video vision transformer. In *IEEE/CVF International Conference on Computer Vision*, 2021.
- [6] Žiga Avsec, Vikram Agarwal, Daniel Visentin, Joseph R. Ledsam, Agnieszka Grabska-Barwinska, Kyle R. Taylor, Yannis Assael, John Jumper, Pushmeet Kohli, and David R. Kelley. Effective gene expression prediction from sequence by integrating long-range interactions. *Nature Methods*, 18(10), 2021.
- [7] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Advances in Neural Information Processing Systems*, 2020.
- [8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.
- [9] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. BEiT: BERT pre-training of image transformers. In *International Conference on Learning Representations*, 2022.

- [10] Mario Bebendorf. *Matrices with Hierarchical Low-Rank Structures*. Lecture Notes in Mathematics. Springer, 2008.
- [11] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [12] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *International Conference on Machine Learning*, 2021.
- [13] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, 2021.
- [14] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurélien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.
- [15] Tom B. Brown et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.
- [16] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Bohnhof, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 2012.
- [17] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 2016.
- [18] Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. Operon c++: an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20*, page 1562–1570. Association for Computing Machinery, 2020. doi: 10.1145/3377929.3398099. URL <https://doi.org/10.1145/3377929.3398099>.
- [19] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, 2020.

- [20] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *IEEE/CVF International Conference on Computer Vision*, 2021.
- [21] William G. La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. In *NeurIPS Datasets and Benchmarks*, 2021. URL <https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c0c7c76d30bd3dcaefc96f40275bdc0a-Abstract-round1.html>.
- [22] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021.
- [23] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *USENIX Conference on Operating Systems Design and Implementation*, 2018.
- [24] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, 2020.
- [25] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.
- [26] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- [27] Krzysztof Marcin Choromanski et al. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.
- [28] Harald Cramér. On the composition of elementary errors. Second paper: Statistical applications. *Scandinavian Actuarial Journal*, 1928.

- [29] Miles Cranmer. Interpretable machine learning for science with PySR and SymbolicRegression.jl. *arXiv preprint arXiv:2305.01582*, 2023. doi: 10.48550/arXiv.2305.01582. URL <https://arxiv.org/abs/2305.01582>.
- [30] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024.
- [31] Tri Dao and Albert Gu. Transformers are SSMS: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024.
- [32] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.
- [33] Stéphane d’Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. ODEFormer: Symbolic regression of dynamical systems with transformers. In *International Conference on Learning Representations*, 2024.
- [34] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2009.
- [35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.
- [36] Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. LongNet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023.
- [37] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. CSWin transformer: A general vision transformer backbone with cross-shaped windows. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.
- [38] Alexey Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

- [39] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020.
- [40] Juan L. Gamella, Jonas Peters, and Peter Bühlmann. Causal chambers as a real-world physical testbed for AI methodology. *Nature Machine Intelligence*, 2025.
- [41] Arya Grayeli, Atharva Seyedsalehi, and Miles Cranmer. Symbolic regression with a learned concept library. In *Advances in Neural Information Processing Systems*, 2024.
- [42] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 1987.
- [43] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 2012.
- [44] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling*, 2024.
- [45] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented transformer for speech recognition. In *Proceedings of Interspeech*, 2020.
- [46] Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. In *International Conference on Learning Representations*, 2021.
- [47] W. Hackbusch and B.N. Khoromskij. A sparse H-matrix arithmetic: general complexity estimates. *Journal of Computational and Applied Mathematics*, 2000.
- [48] Wolfgang Hackbusch. A sparse matrix arithmetic based on H-matrices. part I: Introduction to H-matrices. *Computing*, 1999.
- [49] Ali Hassani, Steven Walton, Jiachen Li, Shen Li, and Humphrey Shi. Neighborhood attention transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

- [50] Baihe He, Qiang Lu, Qingyun Yang, Jake Luo, and Zhiguang Wang. Taylor genetic programming for symbolic regression, 2022. URL <https://arxiv.org/abs/2205.09751>.
- [51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
- [52] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.
- [53] Christina Heinze-Deml, Jonas Peters, and Nicolai Meinshausen. Invariant causal prediction for nonlinear models. *Journal of Causal Inference*, 6(2):20170016, 2018. doi: 10.1515/jci-2017-0016.
- [54] Jonathan Ho and Nal Kalchbrenner. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.
- [55] Samuel Holt, Zhaozhi Qian, and Mihaela van der Schaar. Deep generative symbolic regression, 2023. URL <https://arxiv.org/abs/2401.00282>.
- [56] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=rygGQyrFvH>.
- [57] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [58] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinulescu, and Douglas Eck. Music transformer. In *International Conference on Learning Representations*, 2019.
- [59] Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V. Davuluri. DNABERT: pre-trained bidirectional encoder representations from transformers model for DNA-language in genome. *Bioinformatics*, 37(15), 2021.
- [60] Ying Jin, Weilin Fu, Jian Kang, Jiadong Guo, and Jian Guo. Bayesian symbolic regression, 2020. URL <https://arxiv.org/abs/1910.08892>.

- [61] John Jumper et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 2021.
- [62] Pierre-Alexandre Kamienny, Stéphane d’Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.
- [63] Pierre-Alexandre Kamienny, Stéphane d’Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. In *Advances in Neural Information Processing Systems*, 2022.
- [64] Pierre-Alexandre Kamienny, Guillaume Lample, Sylvain Lamprier, and Marco Virgolin. Deep generative symbolic regression with Monte-Carlo tree search. *International Conference on Machine Learning*, 2023.
- [65] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [66] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, 2020.
- [67] Maarten Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming*, pages 70–82. Springer Berlin Heidelberg, 2003.
- [68] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. In *IEEE/CVF International Conference on Computer Vision*, 2023.
- [69] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.
- [70] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, 2006.
- [71] Andrey Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Giornale dell’Istituto Italiano degli Attuari*, 1933.

- [72] Michael Kommenda, Andreas Beham, Michael Affenzeller, and Gabriel Kronberger. Complexity measures for multi-objective symbolic regression. In *Computer Aided Systems Theory – EUROCAST 2015*, pages 409–416. Springer, 2015. doi: 10.1007/978-3-319-27340-2_51.
- [73] Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. Operon C++: An efficient genetic programming framework for symbolic regression. *Genetic Programming and Evolvable Machines*, 2020.
- [74] Michael F. Korn. *Accuracy in Symbolic Regression*, pages 129–151. Springer New York, 2011. doi: 10.1007/978-1-4614-1770-5_8. URL https://doi.org/10.1007/978-1-4614-1770-5_8.
- [75] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [76] Gabriel Kronberger, Fabrício Olivetti de França, Bogdan Burlacu, Christian Haider, and Michael Kommenda. Shape-constrained symbolic regression—improving extrapolation with prior knowledge. *Evolutionary Computation*, 30(1):75–98, 2022. doi: 10.1162/evco_a_00294.
- [77] Gabriel Kronberger, Fabrício Olivetti de França, Harry Desmond, Deaglan J. Bartlett, and Lukas Kammerer. The inefficiency of genetic programming for symbolic regression. In *Parallel Problem Solving from Nature – PPSN XVIII*, pages 273–289. Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-70055-2_17. URL <https://arxiv.org/abs/2404.17292>.
- [78] David Krueger, Ethan Caballero, Jörn-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Rémi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (REx). In *International Conference on Machine Learning*, 2021.
- [79] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018.
- [80] William La Cava, Lee Spector, and Kourosh Danai. Epsilon-lexicase selection for regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO ’16, pages 741–748. ACM, July 2016. doi: 10.1145/2908812.2908898. URL <http://dx.doi.org/10.1145/2908812.2908898>.

- [81] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. In *NeurIPS Datasets and Benchmarks Track*, 2021.
- [82] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020.
- [83] Mikel Landajuela, Chak Shing Lee, Jiachen Yang, Ruben Glatt, Claudio P Santiago, Ignacio Aravena, Terrell Mundhenk, Garrett Mulcahy, and Brenden K Petersen. A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems*, 35:33985–33998, 2022.
- [84] Heejun Lee, Geon Park, Jaduk Suh, and Sung Ju Hwang. InfiniteHiP: Extending language model context up to 3 million tokens on a single GPU. *arXiv preprint arXiv:2502.08910*, 2025.
- [85] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, 2019.
- [86] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks, 2019. URL <https://arxiv.org/abs/1810.00825>.
- [87] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Annual Meeting of the Association for Computational Linguistics*, 2020.
- [88] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning*, 2022.
- [89] Qian Li, Yuxiao Hu, Juncheng Liu, and Yuntian Chen. GenSR: Symbolic regression based on equation generative space. In *The Fourteenth International Conference on Learning Representations*, 2026. URL <https://openreview.net/forum?id=8emIjwUQZg>.

- [90] Ruiqi Li, Yiting Qi, Shenghe Fang, Jinling Ren, and Xinghua Liu. DiagSWin: A multi-scale vision transformer with diagonal-shaped windows for object detection and segmentation. *Neural Networks*, 2024.
- [91] Wenqiang Li, Weijun Li, Lina Yu, Min Wu, Linjun Sun, Jingyi Liu, Yanjie Li, Shu Wei, Yusong Deng, and Meilan Hao. A neural-guided dynamic symbolic network for exploring mathematical expressions from data, 2024. URL <https://arxiv.org/abs/2309.13705>.
- [92] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics*, 2021.
- [93] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He. Exploring plain vision transformer backbones for object detection. In *European Conference on Computer Vision*, pages 280–296, 2022.
- [94] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, 2014.
- [95] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.
- [96] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. In *International Conference on Learning Representations*, 2018.
- [97] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- [98] Yun Liu, Yu-Huan Wu, Guolei Sun, Le Zhang, Ajad Chhatkuli, and Luc Van Gool. Vision transformers with hierarchical attention. *Machine Intelligence Research*, 2024.
- [99] Ze Liu, Yutong Lin, Yutao Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *IEEE/CVF International Conference on Computer Vision*, 2021.
- [100] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. ViLBERT: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, 2019.

- [101] Matt Mahoney. Large text compression benchmark. <http://www.mattmahoney.net/dc/text.html>, 2011.
- [102] Per-Gunnar Martinsson. Fast multipole methods. In *Encyclopedia of Applied and Computational Mathematics*. Springer, 2015.
- [103] Yoshitomo Matsubara, Naoya Chiba, Ryo Igarashi, and Yoshitaka Ushiku. Rethinking symbolic regression datasets and benchmarks for scientific discovery. *Journal of Data-centric Machine Learning Research*, 2024. URL <https://openreview.net/forum?id=qrUdrXsiXX>.
- [104] Trent McConaghy. Ffx: Fast, scalable, deterministic symbolic regression technology. In *Genetic Programming Theory and Practice IX*, pages 235–260. Springer, 2011.
- [105] Kazem Meidani, Parshin Shojaee, Chandan K. Reddy, and Amir Barati Farimani. SNIP: Bridging mathematical symbolic and numeric realms with unified pre-training. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=KZSEgJGPxu>.
- [106] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.
- [107] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL <https://doi.org/10.7717/peerj-cs.103>.
- [108] Giorgio Morales and John W. Sheppard. Univariate skeleton prediction in multivariate systems using transformers. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2024.
- [109] Terrell N. Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P. Santiago, Daniel faissol, and Brenden K. Petersen. Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL <https://openreview.net/forum?id=tjwQa0I9tdy>.

- [110] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.
- [111] Tan Nguyen, Vai Suliafu, Stanley Osher, Long Chen, and Bao Wang. FMMformer: Efficient and flexible transformer via decomposed near-field and far-field attention. *Advances in Neural Information Processing Systems*, 2021.
- [112] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, 2019.
- [113] Runhai Ouyang, Stefano Curtarolo, Emre Ahmetcik, Matthias Scheffler, and Luca M. Ghiringhelli. SISSO: A compressed-sensing method for identifying the best low-dimensional descriptor in an immensity of offered candidates. *Physical Review Materials*, 2018.
- [114] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009.
- [115] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision*, 2023.
- [116] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random feature attention. In *International Conference on Learning Representations*, 2021.
- [117] Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(5):947–1012, 2016. doi: 10.1111/rssb.12167.
- [118] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press, 2017.
- [119] Brenden K. Petersen, Mikel Landajuela, T. Nathan Mundhenk, Claudio P. Santiago, Soo K. Kim, and Joanne T. Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.

- [120] Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=m5Qsh0kBQG>.
- [121] Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022.
- [122] Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosFormer: Rethinking softmax in attention. In *International Conference on Learning Representations*, 2022.
- [123] Francesco Quinzan, Ashkan Soleymani, Patrick Jaillet, Cristian R. Rojas, and Stefan Bauer. Learning invariant representations under general interventions on the response. In *International Conference on Machine Learning*, 2023.
- [124] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019.
- [125] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.
- [126] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, 2023.
- [127] Alec Radford et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021.
- [128] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

- [129] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.
- [130] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15), 2021.
- [131] Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 2011.
- [132] Dominik Rothenhäusler, Nicolai Meinshausen, Peter Bühlmann, and Jonas Peters. Anchor regression: heterogeneous data meet causality. *Journal of the Royal Statistical Society: Series B*, 2021.
- [133] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 2021.
- [134] Michael S. Ryoo, A. J. Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. TokenLearner: What can 8 learned tokens do for images and videos? In *Advances in Neural Information Processing Systems*, 2021.
- [135] Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts. In *International Conference on Learning Representations*, 2020.
- [136] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 2009.
- [137] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Annual Meeting of the Association for Computational Linguistics*, 2016.
- [138] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.
- [139] Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K. Reddy. Transformer-based planning for symbolic regression. In *Advances in Neural Information Processing Systems*, 2023.

- [140] Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K. Reddy. Transformer-based planning for symbolic regression. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=0rVXQEeFEL>.
- [141] Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. SNIP: Bridging mathematical symbolic and numeric realms with unified pre-training. In *International Conference on Learning Representations*, 2024.
- [142] Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. LLM-SR: Scientific equation discovery via programming with large language models. In *International Conference on Learning Representations*, 2025.
- [143] Parshin Shojaee et al. LLM-SRBench: A new benchmark for scientific equation discovery with LLMs. In *International Conference on Machine Learning*, 2025.
- [144] David Silver et al. Mastering the game of go without human knowledge. *Nature*, 2017.
- [145] N. Smirnov. Table for estimating the goodness of fit of empirical distributions. *Annals of Mathematical Statistics*, 1948.
- [146] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. MIT Press, 2nd edition, 2000.
- [147] Steven H. Strogatz. *Nonlinear Dynamics and Chaos : With Applications to Physics, Biology, Chemistry, and Engineering*. Westview Press, 2nd edition, 2015.
- [148] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.
- [149] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021.
- [150] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 2022.

- [151] Yuan Tian, Wenqi Zhou, Michele Viscione, Hao Dong, David Kammer, and Olga Fink. Interactive symbolic regression with co-design mechanism through offline reinforcement learning. *Nature Communications*, 2025.
- [152] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers and distillation through attention. In *International Conference on Machine Learning*, 2021.
- [153] Ryan T. Tymkow, Benjamin D. Schnapp, Mojtaba Valipour, and Ali Ghodsi. Symbolic-diffusion: Deep learning based symbolic regression with D3PM discrete token diffusion. *arXiv preprint arXiv:2510.07570*, 2025.
- [154] Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020. doi: 10.1126/sciadv.aay2631.
- [155] Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. In *Advances in Neural Information Processing Systems*, 2020.
- [156] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O’Neill, R. I. Mckay, and Edgar Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, June 2011. ISSN 1389-2576. doi: 10.1007/s10710-010-9121-2. URL <https://doi.org/10.1007/s10710-010-9121-2>.
- [157] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. SymbolicGPT: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*, 2021.
- [158] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019. URL <https://arxiv.org/abs/1807.03748>.
- [159] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- [160] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

- [161] Ashish Vaswani, Prajit Ramachandran, Aravind Srinivas, Niki Parmar, Blake Hechtman, and Jonathon Shlens. Scaling local self-attention for parameter efficient visual backbones. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.
- [162] M. Virgolin, T. Alderliesten, C. Witteveen, and P. A. N. Bosman. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary Computation*, 29(2):211–237, 06 2021. ISSN 1063-6560. doi: 10.1162/evco_a_00278. URL https://doi.org/10.1162/evco_a_00278.
- [163] Marco Virgolin and Solon P Pissis. Symbolic regression is NP-hard. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL <https://openreview.net/forum?id=LTiaPxqe2e>.
- [164] Marco Virgolin, Tanja Alderliesten, and Peter A. N. Bosman. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19*, pages 1084–1092. Association for Computing Machinery, 2019. doi: 10.1145/3321707.3321758. URL <https://doi.org/10.1145/3321707.3321758>.
- [165] Ekaterina J. Vladislavleva, Guido F. Smits, and Dick den Hertog. Order of non-linearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2009. doi: 10.1109/TEVC.2008.926486.
- [166] Richard von Mises. Statistik und Wahrheit. *Julius Springer*, 1928.
- [167] Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. Fast transformers with clustered attention. In *Advances in Neural Information Processing Systems*, 2020.
- [168] Ningning Wang, Guobing Gan, Peng Zhang, Shuai Zhang, Junqiu Wei, Qun Liu, and Xin Jiang. ClusterFormer: Neural clustering attention for efficient and effective transformer. In *Annual Meeting of the Association for Computational Linguistics*, 2022.
- [169] Shuohang Wang, Luowei Zhou, Zhe Gan, Yen-Chun Chen, Yuwei Fang, Siqu Sun, Yu Cheng, and Jingjing Liu. ClusterFormer: Clustering-based sparse transformer for question answering. In *Findings of the Association for Computational Linguistics*, 2021.

- [170] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.
- [171] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *IEEE/CVF International Conference on Computer Vision*, 2021.
- [172] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. CvT: Introducing convolutions to vision transformers. In *IEEE/CVF International Conference on Computer Vision*, 2021.
- [173] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [174] Yu-Huan Wu, Yun Liu, Xin Zhan, and Ming-Ming Cheng. Pyramid pooling transformer for scene understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.
- [175] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *Advances in Neural Information Processing Systems*, 2021.
- [176] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, 2020.
- [177] Yilong Xu, Yang Liu, and Hao Sun. Reinforcement symbolic regression machine. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=PJVUWpPnZC>.
- [178] Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal self-attention for local-global interactions in vision transformers. In *Advances in Neural Information Processing Systems*, 2021.
- [179] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning*, 2024.

- [180] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, 2024.
- [181] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025.
- [182] Zihan Yu, Jingtao Ding, Yong Li, and Depeng Jin. Symbolic regression via MDLformer-guided search: From minimizing prediction error to minimizing description length. In *International Conference on Learning Representations*, 2025.
- [183] Zihan Yu, Jingtao Ding, Yong Li, and Depeng Jin. Symbolic regression via MDLformer-guided search: from minimizing prediction error to minimizing description length. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=1jAS7cPAU0>.
- [184] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zihang Jiang, Francis E. H. Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token ViT: Training vision transformers from scratch on ImageNet. In *IEEE/CVF International Conference on Computer Vision*, 2021.
- [185] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, 2017.
- [186] Manzil Zaheer et al. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, 2020.
- [187] Zhanpeng Zeng, Sourav Pal, Jeffery Kline, Glenn M Fung, and Vikas Singh. Multi resolution analysis (MRA) for approximate self-attention. In *International Conference on Machine Learning*, 2022.
- [188] Yu Zhang, Bo Pang, and Cewu Lu. Low-resolution self-attention for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.
- [189] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ADE20K dataset. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

- [190] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115, 2021.
- [191] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2021.
- [192] Zhenhai Zhu and Radu Soricut. H-transformer-1d: Fast one-dimensional hierarchical attention for sequences. In *Annual Meeting of the Association for Computational Linguistics*, 2021.