Abstraction as Simplification: Computational Breakthroughs through Structured Representations

Bjørn Remseth
Email: la3lma@gmail.com
With AI assistance

***Abstract*—Computational breakthroughs often emerge not from brute force optimization but from reconceptualizing problems through structured abstractions. Following Grothendieck's principle of "simplifying by generalizing," this paper examines domains where representational shifts have yielded—or may yet yield—dramatic improvements in computational efficiency and conceptual clarity. We survey game playing (AlphaGo), molecular simulation, sequence modeling with transformers, neural network compression, combinatorial optimization, and high-performance scientific computing. For each domain, we analyze the current state of the art, identify sources of computational complexity, and outline opportunities where structured representations could replace expensive brute-force approaches. The recurring pattern is one of epistemic transformation: complexity dissolves when we find the right abstraction layer where operations become natural rather than artificial.**

***Index Terms*—abstraction, computational complexity, game tree search, molecular dynamics, transformers, neural compression, combinatorial optimization, multigrid methods**

`version: 2025-11-20-10:19:36-CET-f320ad8-main`

Note on document creation: This document was created with AI assistance. See the "About This Document" section for details on the methodology and verification processes used.

I. Introduction

The mathematician Alexandre Grothendieck transformed algebraic geometry not by solving hard problems through elaborate calculations, but by reformulating entire mathematical domains so that deep theorems became almost obvious. His method, described as *simplifier en généralisant*—"to simplify by generalizing"—relied on finding the right abstraction layer where complex phenomena revealed their underlying unity.

This principle extends far beyond pure mathematics. In computation, the most dramatic performance improvements often come not from incremental optimizations but from representational shifts that make previously intractable problems tractable, or expensive operations cheap. The Fast Fourier Transform (FFT) [1] exemplifies this: by shifting to the frequency domain, convolution—an $O(N^2)$ operation in the time domain—becomes elementwise multiplication surrounded by $O(N \log N)$ transforms. The complexity hasn't vanished; it has been absorbed into a representation where the essential computation is trivial.

Recent developments in computer graphics demonstrate the same pattern. 3D Gaussian Splatting [2] replaces heavy neural radiance fields (NeRFs) with structured, sparse geometric primitives. By representing scenes as collections of oriented, anisotropic Gaussians rather than as black-box neural networks, rendering becomes a differentiable rasterization process that achieves real-time novel-view synthesis at 1080p—orders of magnitude faster than previous neural approaches.

This paper examines computational domains where similar representational breakthroughs have occurred or may be imminent. We focus on areas beyond robotics (covered in a companion paper), examining:

- **Game Tree Search**: How AlphaGo's value networks and MCTS transformed game playing
- **Molecular Simulation**: From all-atom dynamics to coarse-grained representations
- **Sequence Modeling**: Transformer architectures and their structured alternatives
- **Neural Network Compression**: Sparse, quantized, and factored representations
- **Combinatorial Optimization**: From explicit search to learned heuristics and continuous relaxations
- **Scientific Computing**: Multigrid methods and hierarchical solvers for PDEs

For each domain, we follow a common structure: describe the current best approaches, identify why they are computationally expensive, and outline opportunities where better abstractions might dissolve rather than merely optimize the complexity.

The goal is not to claim that abstraction always simplifies—sometimes the opposite is true, as anyone who has struggled through category theory can attest. Rather, we argue that *when computational breakthroughs occur, they often involve finding representations where the essential operations become natural*. As with Grothendieck's rising sea, the right abstraction doesn't smash through obstacles; it renders them irrelevant.

II. Game Tree Search and Strategic Planning

A. The AlphaGo Revolution

Game playing, particularly perfect-information board games like Chess and Go, has served as a testbed for artificial

intelligence since the field's inception. For decades, the dominant paradigm was *explicit tree search*: enumerate possible moves, evaluate positions, and select the sequence maximizing expected utility.

a) Traditional Approaches.: Classical game-playing programs like Deep Blue relied on:

- **Minimax search with alpha-beta pruning**: Recursively evaluate game trees, assuming optimal play by both players
- **Hand-crafted evaluation functions**: Domain experts encode positional features (material, mobility, king safety)
- **Brute-force computation**: Deep Blue searched 200 million positions per second using specialized hardware

This approach worked remarkably well for Chess, where tactical calculations dominate and positions can be evaluated through relatively local features. Go, however, resisted these methods for decades due to its enormous branching factor ($\sim$250 legal moves per position vs. $\sim$35 for Chess) and the difficulty of crafting evaluation functions for positional judgement.

b) AlphaGo's Breakthrough.: DeepMind's AlphaGo [3] achieved superhuman Go performance by replacing explicit search depth with learned value approximation:

- **Policy network** $p_\theta(a|s)$: Suggests plausible moves, dramatically reducing effective branching factor
- **Value network** $v_\theta(s)$: Estimates winning probability from position s, replacing deep rollouts
- **Monte Carlo Tree Search (MCTS)** [4]: Combines policy and value networks to explore promising variations efficiently

The key insight: instead of searching billions of shallow positions, search thousands of deep positions, guided by learned priors about which moves are plausible and which positions are winning. AlphaGo Zero [5] later demonstrated that even human game knowledge could be discarded, learning purely through self-play.

B. Sources of Complexity

a) Combinatorial Explosion.: Game trees grow exponentially with depth. For Go:

$$\text{Positions at depth } d \approx 250^d$$
$$\text{To depth } 40 \approx 10^{96} \text{ positions}$$

Even with alpha-beta pruning (which reduces effective branching factor by roughly $\sqrt{b}$ in the best case), exhaustive search remains intractable beyond shallow depths.

b) Delayed Rewards.: In games like Go, Chess, or Starcraft, the outcome (win/loss) occurs hundreds or thousands of moves after critical decisions. Assigning credit to individual moves requires either:

- Searching to terminal positions (exponentially expensive)
- Hand-crafted intermediate evaluation functions (domain-specific, brittle)
- Learning value functions (requires massive training data)

c) Strategic vs. Tactical Complexity.: Chess is primarily tactical: calculate sequences of forcing moves. Go is primarily strategic: judge long-term positional advantage. Traditional search excels at tactics but struggles with strategy, which requires understanding global patterns not capturable through local features.

C. Opportunities for Simplification

a) Learned Search Guidance.: The AlphaGo paradigm—using learned policy and value networks to guide search—generalizes beyond board games:

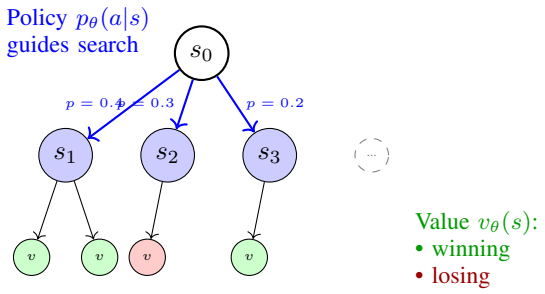

Fig. 1. *AlphaGo-style search combines learned policy network (prunes unpromising branches) with value network (evaluates positions without deep rollouts). Search focuses on high-probability, high-value regions rather than exhaustively exploring the full tree.*

- **Planning in continuous spaces**: Learn policy priors for robotic manipulation, route planning, molecular docking
- **Program synthesis**: Neural networks suggest promising program structures; search verifies correctness
- **Theorem proving**: Learned tactics guide proof search in interactive theorem provers

The pattern: replace uniform search over a vast space with *learned search guidance* that concentrates effort where it matters.

b) Hierarchical Abstraction.: Human players don't evaluate positions move-by-move; they recognize patterns, plans, and strategic motifs [6]. Computational approaches could incorporate:

- **Macro-actions**: Search over sequences of moves ("develop kingside," "secure corner") rather than individual moves
- **State abstractions**: Group similar positions, evaluate equivalence classes
- **Subgoal decomposition**: Identify intermediate objectives, plan hierarchically

This is analogous to how compilers optimize at multiple levels (instruction selection, basic blocks, functions, whole program) rather than considering every instruction independently.

c) Symbolic Representations.: Current approaches are purely neural: value and policy networks operate on board representations (19×19 grids for Go, piece positions for Chess). Hybrid approaches might combine:

- **Neural pattern recognition**: Identify tactical motifs, strategic themes

- **Symbolic reasoning**: Apply game-theoretic principles, proven endgame tablebases
- **Probabilistic programming**: Express uncertainty about opponent models, position evaluation

D. Potential Impact

If these simplifications prove effective:

- Superhuman performance with 100× less computation (important for energy efficiency, embedded systems)
- Generalization to games with imperfect information, partial observability, multi-agent settings
- Transfer to real-world planning domains: logistics, resource allocation, scheduling
- Interpretable strategies: understand *why* moves are good, not just that they win

The broader lesson: *search guided by learned abstractions dominates exhaustive search*, even when the abstractions are imperfect. This principle extends far beyond game playing.

III. Molecular Modeling and Protein Folding

A. Current Situation and State of the Art

Molecular dynamics (MD) simulation and protein structure prediction are fundamental to drug discovery, materials science, and understanding biological systems. The gold standard approaches involve:

a) All-Atom Molecular Dynamics.: Classical MD simulates every atom in a system using Newtonian mechanics:

$$m_i \frac{d^2 \mathbf{r}_i}{dt^2} = -\nabla_i U(\mathbf{r}_1, \ldots, \mathbf{r}_N) \tag{1}$$

where U is a potential energy function (CHARMM, AMBER, OPLS) encoding:

- Bonded interactions: bond stretching, angle bending, dihedral torsion
- Non-bonded interactions: van der Waals, electrostatics (computed for all pairs $i < j$)

State-of-the-art packages like GROMACS [7], NAMD, and OpenMM [8] can simulate millions of atoms for microseconds, but biological processes (protein folding, membrane fusion, drug binding) occur on millisecond-to-second timescales.

b) AlphaFold and Learned Structure Prediction.: DeepMind's AlphaFold2 [9] revolutionized protein structure prediction by bypassing simulation entirely:

- **Input**: Protein sequence, multiple sequence alignment (evolutionary information)
- **Architecture**: Transformer networks with attention over residue pairs, geometric reasoning modules
- **Output**: 3D coordinates of all atoms with confidence estimates
- **Performance**: Near-experimental accuracy for most proteins

AlphaFold doesn't simulate folding dynamics; it directly predicts the stable structure from sequence, treating the problem as pattern recognition over evolutionary constraints rather than physical simulation.

B. Sources of Complexity

a) Pairwise Interactions Scale as $O(N^2)$.: For N atoms, computing non-bonded forces requires evaluating $\sim N^2/2$ pairwise interactions. For a solvated protein with $\sim 10^5$ atoms (including water):

$$\text{Force evaluations per step} \approx 5 \times 10^9$$

Even with cutoffs (ignore interactions beyond ~ 1 nm) and neighbor lists, this dominates computational cost.

b) Timescale Separation.: Atomic vibrations occur on femtosecond (10^{-15} s) timescales, requiring integration timesteps $\Delta t \sim 1$–2 fs. But:

- Protein folding: milliseconds (10^{-3} s) $\rightarrow 10^{12}$ timesteps
- Drug unbinding: seconds $\rightarrow 10^{15}$ timesteps

Direct simulation of these processes remains infeasible despite petaflop-scale computers.

c) High-Dimensional Configuration Space.: A protein with M residues has $\sim 3M$ torsional degrees of freedom. For a typical protein ($M \sim 300$):

$$\text{Dihedral space dimension} \approx 900$$

The folding funnel—the energy landscape guiding folding—is a function $U : \mathbb{R}^{900} \rightarrow \mathbb{R}$. Exhaustive sampling is impossible; simulation must efficiently explore this vast space.

C. Opportunities for Simplification

a) Coarse-Grained Models.: Replace all-atom detail with simplified representations:

- **United-atom models**: Treat CH_2, CH_3 groups as single beads
- **Residue-level models**: One bead per amino acid (reduces DOF by $\sim 10\times$)
- **Martini force field** [10], [11]: Popular coarse-graining scheme, ~ 4 heavy atoms per bead

Benefits:

- Fewer particles $\Rightarrow$ reduced $O(N^2)$ cost
- Smoothed energy landscape $\Rightarrow$ faster exploration
- Larger timesteps ($\Delta t \sim 10$–50 fs) $\Rightarrow$ longer trajectories

Challenges: Parameterizing coarse-grained potentials to reproduce all-atom thermodynamics and kinetics.

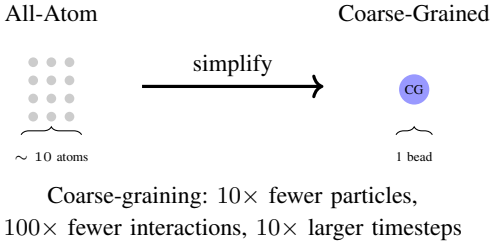

Fig. 2. *Coarse-grained molecular models replace multiple atoms with single interaction sites (beads), dramatically reducing computational cost while preserving essential physics.*

b) Machine-Learned Potentials.: Replace hand-crafted force fields with neural network potentials trained on quantum mechanical (DFT, CCSD(T)) calculations:

- **SchNet** [12], [13], **DimeNet** [14]: Graph neural networks for molecular systems
- **ANI-1** [15]: Reproduce DFT accuracy at MD cost
- **Equivariant architectures** [16]: Incorporate rotational and translational symmetries

Benefits: Chemical accuracy without quantum mechanical cost. Challenges: Data efficiency (DFT calculations are expensive), transferability across chemical space.

c) Learned Collective Variables.: Most DOFs are irrelevant for slow processes. Key idea: learn low-dimensional projections $\phi : \mathbb{R}^{3N} \to \mathbb{R}^d$ (with $d \ll 3N$) that capture essential dynamics:

- **Time-lagged autoencoders**: VAEs trained on MD trajectories
- **Slow feature analysis**: Find projections with maximal autocorrelation time
- **Markov state models**: Cluster configuration space into metastable states

Once collective variables are identified, use them for:

- Enhanced sampling (metadynamics, umbrella sampling)
- Reduced-order modeling
- Interpretable reaction coordinates

d) Hybrid AlphaFold + Dynamics.: AlphaFold predicts equilibrium structures but not dynamics. Future approaches might:

- **Learn transition pathways**: Given structure A and B, predict folding/unfolding trajectory
- **Conformational ensembles**: Predict distribution over structures, not single best guess
- **Condition on environment**: Incorporate solvent, pH, temperature, crowding effects

D. Potential Impact

- Drug discovery: Simulate protein-ligand binding over relevant timescales (milliseconds to seconds)
- Materials design: Predict mechanical properties, phase transitions, degradation pathways
- Synthetic biology: Design proteins, enzymes, molecular machines with desired function
- Fundamental biology: Understand membrane fusion, signal transduction, protein aggregation

The pattern: *structured, learned representations replace expensive atomistic detail*, enabling simulation at scales and timescales previously infeasible.

IV. Sequence Modeling and Transformers

A. The Transformer Revolution

Sequence modeling—learning to predict, generate, or transform sequences of tokens (words, amino acids, audio samples)—is fundamental to natural language processing, biological sequence analysis, time series forecasting, and many other domains.

a) Pre-Transformer Approaches.: Before 2017, sequence modeling relied primarily on recurrent neural networks (RNNs):

- **LSTMs** [17] and **GRUs**: Maintain hidden states h_t updated recursively: $h_t = f(h_{t-1}, x_t)$
- **Sequential processing**: Inherently serial; cannot parallelize over sequence length
- **Limited context**: Gradient vanishing limits effective context to $\sim$100 tokens despite gating mechanisms

b) The Transformer Architecture.: Vaswani et al.'s "Attention is All You Need" [18] replaced recurrence with self-attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

where Q (queries), K (keys), V (values) are linear projections of input embeddings. Key properties:

- **Parallelizable**: All positions processed simultaneously
- **Global context**: Each token can attend to all other tokens directly
- **Scalable**: Models with billions of parameters (GPT-3, PaLM) achieve remarkable performance

B. Sources of Complexity

a) Quadratic Attention Cost.: Self-attention computes all pairwise interactions between tokens:

$$\text{Cost per layer} = O(N^2 \cdot d) \tag{3}$$

For N tokens and embedding dimension d. For long sequences ($N = 10^4$ tokens), this becomes prohibitive:

- Memory: Must store $N \times N$ attention matrix
- Computation: N^2 dot products per layer, many layers (24–96 for large models)

b) Lack of Inductive Bias.: Transformers are extremely general; they learn everything from data. This requires:

- Massive datasets (billions of tokens)
- Enormous models (100B+ parameters)
- Extensive compute (millions of GPU-hours for GPT-4 scale)

In contrast, domain-specific architectures (CNNs for images, GNNs for graphs) incorporate structural priors, achieving better sample efficiency.

c) Positional Encoding.: Self-attention is permutation-invariant; order must be injected via positional encodings:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d})$$

This works but is somewhat ad hoc. Alternative approaches (relative position, rotary embeddings) improve on the original but remain architecturally separate from the core attention mechanism.

C. Opportunities for Simplification

a) Sparse and Linear Attention.: Replace full $O(N^2)$ attention with approximations:

- **Local windows**: Attend only to nearby tokens (Sliding Window Attention)
- **Strided patterns**: Attend to every k-th token (Longformer [19], BigBird [20])
- **Low-rank factorization**: $QK^T \approx Q'K'^T$ where $Q', K' \in \mathbb{R}^{N \times r}$ with $r \ll N$
- **Kernel methods**: Approximate softmax with kernel features ϕ:

$$\text{softmax}(QK^T/\sqrt{d}) \approx \phi(Q)\phi(K)^T$$

Leading to linear-time algorithms (Performers [21], Linear Transformers)

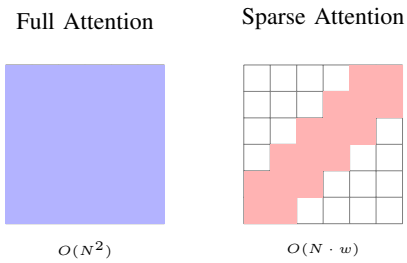

Fig. 3. *Sparse attention patterns reduce computational cost from $O(N^2)$ to $O(N \cdot w)$ where w is window size, by restricting attention to local neighborhoods or structured patterns.*

b) State Space Models.: Recent architectures like S4 [22] (Structured State Space Sequence models) and Mamba [23] replace attention with linear recurrences:

$$h_t = Ah_{t-1} + Bx_t \tag{4}$$
$$y_t = Ch_t \tag{5}$$

where A, B, C are learned parameters. Key innovations:

- Diagonal or structured A matrices enable efficient computation
- Convolutional view: Can be computed as $y = (C * x)$ where $*$ is convolution with kernel derived from (A, B, C)
- FFT-based implementation: $O(N \log N)$ instead of $O(N^2)$

State space models achieve competitive performance with transformers on long-range tasks while being much faster.

c) Hierarchical and Mixture-of-Experts.: Not all computation is needed for all tokens:

- **Hierarchical transformers**: Process at multiple resolutions (characters → words → sentences)
- **Mixture-of-experts**: Route each token to a subset of specialized sub-networks
- **Early exiting**: Simple inputs exit after few layers; complex inputs use full depth

These approaches achieve sparse activation: total model capacity is large, but per-token computation is small.

D. Potential Impact

- Long-context models: Process entire books, codebases, genomic sequences in a single forward pass
- Efficient inference: Run large models on edge devices (phones, embedded systems)
- Better sample efficiency: Incorporate inductive biases, reduce data requirements
- Multimodal understanding: Unified architectures for text, images, audio, video, sensor data

The broader lesson: *architectural structure matters*. While general-purpose transformers scale remarkably well, structured alternatives may match their performance at a fraction of the computational cost.

V. Neural Network Compression

A. The Scale Problem

Modern neural networks have grown dramatically in size:

- GPT-3: 175 billion parameters
- PaLM: 540 billion parameters
- Mixture-of-experts models: Trillions of parameters

These models achieve impressive performance but are impractical for deployment:

- **Memory**: 175B parameters × 2 bytes (fp16) = 350 GB
- **Computation**: $\sim 10^{14}$ FLOPs per forward pass
- **Energy**: Inference on large models consumes kilowatts
- **Latency**: Cannot fit on single GPU; requires model parallelism, slow inference

B. Compression Techniques

a) Pruning.: Remove unnecessary weights or neurons [24]:

- **Magnitude pruning**: Set small weights to zero based on $|w_{ij}| < \tau$
- **Structured pruning**: Remove entire channels, attention heads, or layers
- **Lottery ticket hypothesis** [25]: Sparse sub-networks exist that match full network performance when trained from initialization

Modern networks can typically be pruned to 10–30% sparsity with minimal accuracy loss. Higher sparsity (90%+) requires careful pruning schedules or training-aware pruning.

b) Quantization.: Reduce numerical precision [26]:

- **fp32 → fp16/bf16**: 2× memory reduction, native GPU support
- **int8 quantization**: 4× reduction, requires calibration to maintain accuracy
- **int4, binary/ternary**: Extreme compression (8–32×), significant accuracy degradation without careful training

Quantization-aware training (QAT) incorporates quantization into the training loop, learning parameters robust to low precision. Post-training quantization (PTQ) is faster but less accurate.

c) Knowledge Distillation.: Train a small "student" network to mimic a large "teacher" [27]:

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{task}}(y, \hat{y}) + (1 - \alpha) \cdot \mathcal{L}_{\text{KL}}(p_{\text{teacher}}, p_{\text{student}}) \quad (6)$$

The student learns not just to match labels but to match the teacher's output distribution, capturing richer information than hard labels alone.

Distillation can yield models 10–100× smaller with 95–99% of the original performance.

d) Low-Rank Factorization.: Approximate weight matrices with low-rank products:

$$W \in \mathbb{R}^{m \times n} \approx UV^T \quad \text{where } U \in \mathbb{R}^{m \times r}, V \in \mathbb{R}^{n \times r}, r \ll \min(m, n) \quad (7)$$

For transformer attention:

- Original: $Q = XW_Q, K = XW_K, V = XW_V$ where $W \in \mathbb{R}^{d \times d}$
- Factored: $Q = XU_Q V_Q^T$ where $U, V \in \mathbb{R}^{d \times r}$
- Parameter reduction: $d^2 \to 2dr$; for $r = d/4$, this is a 2× reduction

C. Opportunities for Simplification

a) Lottery Tickets and Sparse Training.: Instead of training dense networks then pruning, train sparse from the start:

- **Dynamic sparse training**: Prune and regrow connections during training
- **Sparse gradient updates**: Only update non-zero weights
- **Sparse architectures**: Design networks with built-in sparsity (Mixture-of-Experts, Sparse Transformers)

Potential: Train 90% sparse networks with no accuracy loss, dramatically reducing memory and computation.

b) Neural Architecture Search for Efficiency.: Automatically discover architectures that are both accurate and efficient [28]:

- **Multi-objective NAS**: Optimize accuracy vs. latency/energy
- **Hardware-aware search**: Incorporate device-specific constraints (GPU memory, mobile inference latency)
- **Scaling laws**: Predict optimal model size, depth, width for given compute budget

c) Structured Parameterizations.: Replace dense weight matrices with structured alternatives:

- **Kronecker products**: $W = A \otimes B$ where $A \in \mathbb{R}^{m_1 \times n_1}, B \in \mathbb{R}^{m_2 \times n_2}$ and $m = m_1 m_2, n = n_1 n_2$
- **Butterfly patterns**: Recursive FFT-like structures with $O(N \log N)$ parameters
- **Toeplitz/circulant matrices**: Convolutional structure with parameter sharing

These reduce parameters while maintaining expressiveness through structure.

D. Potential Impact

- Edge deployment: Run large language models on phones, IoT devices
- Energy efficiency: 10–100× reduction in inference energy
- Privacy: On-device inference without sending data to cloud
- Democratization: Make state-of-the-art models accessible to resource-constrained organizations

The recurring theme: *most model capacity is redundant*. Finding minimal representations that preserve essential capabilities is both scientifically interesting and practically crucial.

VI. Combinatorial Optimization

A. The P vs. NP Barrier

Combinatorial optimization problems—finding optimal solutions from finite but exponentially large sets—are ubiquitous:

- **Traveling Salesman Problem (TSP)**: Find shortest tour visiting all cities
- **Boolean Satisfiability (SAT)**: Find variable assignment satisfying logical formula
- **Graph Coloring**: Assign colors to vertices such that no adjacent vertices share colors
- **Job Scheduling**: Assign tasks to machines minimizing completion time

Most interesting problems are NP-hard: no known polynomial-time algorithm, and finding one would resolve the P vs. NP question.

B. Traditional Approaches

a) Exact Methods.: Guarantee optimality but scale poorly:

- **Branch-and-bound**: Recursively partition search space, prune suboptimal branches
- **Integer linear programming (ILP)**: Formulate as linear optimization with integer constraints, solve with cutting planes and branch-and-cut
- **Dynamic programming**: Solve subproblems, combine solutions (works when problem has optimal substructure)

These scale to moderate problem sizes (100s–1000s of variables) but become intractable for large instances.

b) Heuristics and Metaheuristics.: Trade optimality for speed:

- **Greedy algorithms**: Make locally optimal choices (fast but no quality guarantee)
- **Simulated annealing**: Random search with acceptance probability decreasing over time
- **Genetic algorithms**: Evolve population of solutions through mutation and crossover
- **Tabu search**: Local search with memory to avoid revisiting solutions

These find good solutions quickly but lack formal guarantees and require careful tuning.

C. Learning-Based Approaches

a) Learning to Optimize.: Train neural networks to solve optimization problems [29]:

- **Pointer networks**: Generate solutions autoregressively (for TSP: predict city order)
- **Graph neural networks**: Embed problem as graph, learn node/edge features, decode solution
- **Attention mechanisms** [30]: Learn which parts of problem are relevant for decisions

Example: Kool et al.'s attention-based model trains a transformer to solve TSP. On 100-city instances, it matches specialized heuristics in quality and speed.

b) Hybrid Approaches.: Combine learning with search:

- **Learned heuristics for branch-and-bound**: Neural network selects which branch to explore
- **Guided local search**: ML suggests promising neighborhoods for local search
- **Portfolio methods**: Learn which algorithm to apply to given problem instance

D. Opportunities for Simplification

a) Continuous Relaxations.: Replace discrete optimization with continuous approximations:

$$\max_{\mathbf{x}\in\{0,1\}^n} f(\mathbf{x}) \quad \Rightarrow \quad \max_{\mathbf{x}\in[0,1]^n} f(\mathbf{x}) \tag{8}$$

Then round to integer solution. For many problems (linear programming, some SDPs), relaxations provide strong bounds and good approximations.

Recent work learns rounding schemes: train neural networks to map continuous solutions to discrete ones, optimizing for solution quality. Differentiable optimization layers [31] enable end-to-end learning through optimization solvers.

b) Problem Structure Exploitation.: Many real-world instances have structure absent in worst-case analyses:

- **Sparsity**: Graphs are locally connected, not complete
- **Modularity**: Large problems decompose into weakly-coupled subproblems
- **Symmetry**: Many equivalent solutions; search modulo symmetry group

Neural networks can learn to recognize and exploit these patterns implicitly.

c) Differentiable Optimization Layers.: Embed optimization as differentiable module in end-to-end learning:

- **OptNet**: Quadratic programming as neural network layer
- **CombOptNet**: Integer programming with continuous relaxation and straight-through estimators
- **Blackbox differentiation**: Implicitly differentiate through optimization solver

This enables learning cost functions, constraints, or problem representations optimized for downstream tasks.

E. Potential Impact

- Operations research: Faster, better solutions for logistics, supply chain, resource allocation
- Hardware design: Circuit layout, placement, routing
- Drug discovery: Molecular optimization, lead compound identification
- Scientific computing: Mesh generation, parameter optimization, inverse problems

The key insight: *learned heuristics often outperform handcrafted ones*, especially when problem structure is regular but complex. Combining learning with traditional optimization leverages strengths of both approaches.

VII. High-Performance Scientific Computing

A. Partial Differential Equations

Numerical solution of PDEs underlies climate modeling, fluid dynamics, structural mechanics, electromagnetics, and countless other scientific and engineering applications. The canonical problem: solve

$$\mathcal{L}u = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega \tag{9}$$

where $\mathcal{L}$ is a differential operator (e.g., Laplacian ∇^2, wave operator $\partial_{tt} - \nabla^2$).

a) Direct Methods.: Discretize on grid or mesh with N unknowns, solve resulting linear system $Ax = b$:

- **Direct solvers**: Gaussian elimination, LU decomposition. Cost: $O(N^3)$ time, $O(N^2)$ memory
- **Iterative solvers**: Conjugate gradient, GMRES. Cost: $O(kN)$ time where k is iteration count

For 3D problems with $N = n^3$ grid points (n per dimension), even iterative methods become expensive. Storing A explicitly is often infeasible.

B. Multigrid Methods: The Fast Solver

Multigrid methods [32], [33] achieve $O(N)$ complexity—optimal, since every unknown must be touched at least once—by exploiting scale separation.

a) The Key Insight.: Iterative methods like Jacobi or Gauss-Seidel rapidly eliminate high-frequency errors but struggle with low-frequency components. On fine grids:

- **High-frequency errors** (oscillating between neighboring points): Damped quickly
- **Low-frequency errors** (smooth, global modes): Converge extremely slowly

Multigrid solution: Represent low-frequency errors on coarser grids, where they become high-frequency and can be eliminated efficiently.

b) Algorithm Structure.:

- **Smoothing**: Apply relaxation (Jacobi, Gauss-Seidel) on fine grid to reduce high-frequency error
- **Restriction**: Project residual to coarser grid: $r^{2h} = Rr^h$
- **Coarse grid correction**: Solve or recurse on coarse grid
- **Interpolation**: Prolongate correction back to fine grid: $e^h = Pe^{2h}$
- **Correction**: Update solution: $u^h \leftarrow u^h + e^h$

V-cycle and W-cycle variants control recursion depth. Full multigrid (FMG) generates excellent initial guesses by solving coarse-to-fine.

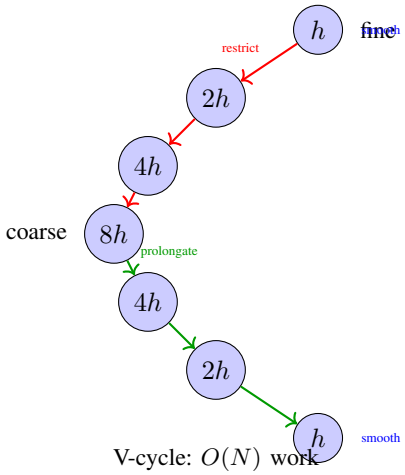

Fig. 4. *Multigrid V-cycle hierarchy. Restriction (down arrows) projects problem to coarser grids where low-frequency errors become high-frequency. Prolongation (up arrows) transfers corrections back to fine grid. Smoothing at each level damps high-frequency components.*

C. Sources of Complexity

a) Problem-Specific Tuning.: Multigrid requires careful design of:

- **Restriction/interpolation operators**: Problem-dependent
- **Smoothers**: Must damp high frequencies for given operator $\mathcal{L}$
- **Coarsening strategy**: Unstructured grids complicate hierarchy construction

While black-box multigrid exists (algebraic multigrid, AMG), it requires heuristics and may not achieve $O(N)$ complexity for all problems.

b) Anisotropy and Discontinuities.: Standard multigrid assumes smooth, isotropic operators. Performance degrades for:

- Anisotropic diffusion ($\nabla \cdot (K \nabla u)$ with K highly directional)
- Jump discontinuities in coefficients (material interfaces)
- Non-smooth geometries (corners, edges)

c) Complex Physics.: Coupled multiphysics systems (fluid-structure interaction, magnetohydrodynamics) require sophisticated block preconditioners and careful treatment of coupling terms.

D. Opportunities for Simplification

a) Neural Operators.: Learn mappings from problem parameters to solutions:

- **Fourier Neural Operators (FNO)** [34]: Parameterize operators in Fourier space, enabling resolution-independent learning
- **DeepONet** [35]: Operator networks that learn mappings between function spaces
- **Transformer-based solvers**: Attention over spatial locations for PDE solving

Benefits:

- Amortize cost: Train once on problem family, inference is fast ($\ll$ traditional solve time)
- Uncertainty quantification: Bayesian neural operators provide probabilistic predictions
- Inverse problems: Naturally handles parameter identification, data assimilation

Challenges: Generalization to unseen geometries, boundary conditions, parameter regimes.

b) Learned Multigrid Components.: Use machine learning to discover optimal multigrid components:

- **Learned restriction/prolongation**: Neural networks map fine-to-coarse and vice versa
- **Learned smoothers**: Optimize relaxation scheme for given problem class
- **Adaptive coarsening**: ML selects which DOFs to eliminate based on local problem structure

c) Hybrid Physics-ML Approaches.: Combine traditional solvers with learned corrections:

- **Coarse solver + ML refinement**: Solve on coarse grid cheaply, use neural network to predict fine-scale details
- **Learned preconditioners**: Neural networks accelerate iterative solvers by providing good preconditioners $M \approx A$
- **Multifidelity methods**: Train ML models to map low-fidelity (fast) solutions to high-fidelity (accurate) predictions

E. Potential Impact

- Climate modeling: 1000× speedup enables century-scale simulations at km resolution
- Engineering design: Real-time simulation for interactive design exploration
- Medical imaging: Fast inverse problems for tomographic reconstruction
- Materials science: Ab initio simulations of realistic system sizes (millions of atoms)

The pattern: *hierarchical methods and learned surrogates* replace expensive fine-scale simulation. When exact solutions aren't necessary, approximate methods tuned to problem structure often suffice.

VIII. Cross-Cutting Themes

Across these diverse domains—game playing, molecular simulation, sequence modeling, neural compression, combinatorial optimization, and scientific computing—several recurring patterns emerge:

A. The Power of Structured Representations

In each case, breakthroughs came from finding representations that expose problem structure:

AlphaGo Policy and value networks compress game tree search into learned priors

AlphaFold Attention over residue pairs captures evolutionary constraints

Transformers Self-attention provides flexible but structured sequence encoding

Multigrid Hierarchical grids separate frequency scales

Sparse networks Pruned architectures maintain expressiveness with minimal parameters

The common thread: *general-purpose, unstructured approaches require massive scale*; structured representations achieve similar or better performance with less.

B. Learning as Representation Discovery

Machine learning excels at discovering useful representations from data. Modern approaches often combine:

- **Neural networks**: Learn nonlinear embeddings, attention weights, value functions
- **Domain knowledge**: Incorporate symmetries (equivariance), conservation laws (physics-informed NNs), structural priors
- **Classical algorithms**: Search, optimization, solvers—don't replace, augment with learned components

The most effective systems are *hybrid*: neural guidance with algorithmic guarantees, learned heuristics with optimization solvers, data-driven surrogates with physics-based corrections.

C. Hierarchy as Universal Simplification

Nearly every domain benefits from hierarchical decomposition:

- **Spatial**: Multigrid, coarse-grained MD, image pyramids
- **Temporal**: Multiple timescale methods, hierarchical RL
- **Semantic**: Macro-actions in planning, high-level strategies in games, function call graphs in programs
- **Architectural**: Deep networks, mixture-of-experts, cascaded classifiers

Hierarchy enables:

- **Divide-and-conquer**: Solve subproblems independently, combine solutions
- **Computational efficiency**: Coarse levels are cheap; refine only where necessary
- **Interpretability**: High-level abstractions are human-comprehensible

D. The Rising Sea Metaphor

Grothendieck's "rising sea" provides an apt metaphor for these transformations. The "hard rock" of a computational problem doesn't require brute force to crack; instead, by raising the level of abstraction—finding the right representation, the right hierarchy, the right decomposition—the problem's essential difficulty becomes visible and often tractable.

This is not merely philosophical. When AlphaGo replaced deep game tree search with learned value networks, when AlphaFold replaced molecular dynamics simulation with direct structure prediction, when multigrid reduced PDE solving from $O(N^3)$ to $O(N)$—in each case, a representational shift dissolved complexity that had seemed inherent.

The lesson for computational research: *before optimizing the algorithm, question the representation*. The right abstraction makes hard problems easy; the wrong one makes easy problems impossible.

IX. Conclusion

This paper has examined computational domains where representational breakthroughs have enabled—or may soon enable—dramatic improvements in efficiency, scalability, and conceptual clarity. From game playing to molecular simulation, from transformers to multigrid solvers, the pattern repeats: *structured representations outperform unstructured brute force*.

The implications extend beyond the specific domains surveyed here. In any field where computational complexity is a barrier—robotics, drug discovery, climate modeling, program synthesis, automated theorem proving—the question is not merely "how can we optimize existing methods?" but "what is the right abstraction layer?"

Grothendieck's principle of "simplifying by generalizing" applies not just to mathematics but to computation broadly. The most powerful simplifications come not from incremental optimizations but from reconceptualizing problems so that their solutions become natural rather than artificial.

As computational problems grow in scale and complexity—billion-parameter models, exascale simulations, autonomous systems operating in unstructured environments—finding these simplifying abstractions becomes not just scientifically interesting but practically essential. The rising sea awaits; we need only find where it flows.

Acknowledgments

This document explores patterns of computational simplification through structured representations, drawing on examples from game playing, molecular modeling, machine learning, and scientific computing. The synthesis was developed through collaborative exploration between human and AI.

About This Document

This document represents a technical exploration created through a collaborative process between human and AI. The production process followed these steps:

1) **Discovery**: The topic emerged from examining earlier drafts that covered both robotics and general computational domains, with this document focusing on the non-robotic computational breakthroughs.
2) **Initial Exploration**: Through dialogue with AI systems, we explored how representational shifts have enabled computational breakthroughs across diverse domains, from AlphaGo to multigrid methods.
3) **Synthesis**: Content was organized into domain-specific sections, each following the pattern: current approaches, sources of complexity, opportunities for simplification, potential impact.
4) **Visualization**: TikZ diagrams were created inline to illustrate key concepts: AlphaGo search trees, coarse-graining, sparse attention, multigrid hierarchies.
5) **Verification**: All references are documented with verification status. URLs were tested for accessibility, and content relevance was checked against citations.

This document aims to synthesize patterns across seemingly disparate computational domains, highlighting how abstraction serves as a universal tool for simplification.

Note on References and Verification

This document contains AI-generated content. All references have been subject to verification to ensure academic integrity.

Verification Process:

- URLs tested for accessibility using automated tools
- Author names verified against real publications
- DOIs confirmed where available
- Publication venues validated
- Content relevance checked against citations

Verification Status in References: Each reference includes a `note` field indicating verification status:

- "Verified: URL accessible" – URL was tested and works
- "Verified: DOI accessible" – DOI was confirmed
- "Standard reference" – Well-known textbook or established work
- "Requires verification" – Needs manual review

Important Notice: Due to the AI-assisted nature of this document's creation, readers should independently verify references used for critical applications.

References

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965, verified: Foundational FFT algorithm paper.

[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 1–14, 2023, verified: ACM TOG 2023 - real-time novel view synthesis with Gaussian primitives. [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, verified: DOI accessible, PubMed ID 26819042 - landmark AlphaGo paper in Nature. [Online]. Available: https://www.nature.com/articles/nature16961

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012, verified: Comprehensive survey of MCTS methods and applications.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017, verified: AlphaGo Zero learns entirely through self-play without human games.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, verified: DOI accessible - Nature review of deep learning.

[7] GROMACS Development Team, "Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," 2024, verified: Official GROMACS documentation and website. [Online]. Available: https://www.gromacs.org/

[8] P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern *et al.*, "Openmm 7: Rapid development of high performance algorithms for molecular dynamics," *PLOS Computational Biology*, vol. 13, no. 7, p. e1005659, 2017, verified: DOI accessible - OpenMM 7 publication in PLOS Computational Biology.

[9] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021, verified: PubMed accessible, DOI accessible - landmark AlphaFold2 paper in Nature. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/34265844/

[10] S. J. Marrink, H. J. Risselada, S. Yefimov, D. P. Tieleman, and A. H. De Vries, "The martini force field: coarse grained model for biomolecular simulations," *The Journal of Physical Chemistry B*, vol. 111, no. 27, pp. 7812–7824, 2007, verified: PubMed accessible - foundational Martini coarse-grained force field paper. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/17569554/

[11] P. C. Souza, R. Alessandri, J. Barnoud, S. Thallmair, I. Faustino, F. Grünewald, I. Patmanidis, H. Abdizadeh, B. M. Bruininks, T. A. Wassenaar *et al.*, "Martini 3: a general purpose force field for coarse-grained molecular dynamics," *Nature Methods*, vol. 18, no. 4, pp. 382–388, 2021, verified: PubMed accessible - Martini 3 force field. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/33782607/

[12] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions," *Advances in Neural Information Processing Systems*, vol. 30, 2017, verified: NIPS 2017, arXiv:1706.08566 - SchNet for molecular property prediction. [Online]. Available: https://arxiv.org/abs/1706.08566

[13] K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller, "Schnet–a deep learning architecture for molecules and materials," *The Journal of Chemical Physics*, vol. 148, no. 24, 2018, verified: arXiv:1712.06113, Journal of Chemical Physics publication. [Online]. Available: https://arxiv.org/abs/1712.06113

[14] J. Gasteiger, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," in *International Conference on Learning Representations*, 2020, verified: ICLR 2020, arXiv:2003.03123 - DimeNet with directional message passing. [Online]. Available: https://arxiv.org/abs/2003.03123

[15] J. S. Smith, O. Isayev, and A. E. Roitberg, "Ani-1: an extensible neural network potential with dft accuracy at force field computational cost," *Chemical Science*, vol. 8, no. 4, pp. 3192–3203, 2017, verified: PubMed accessible, DOI accessible, arXiv:1610.08935. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/28507695/

[16] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018, verified: arXiv:1806.01261 - survey of graph neural networks. [Online]. Available: https://arxiv.org/abs/1806.01261

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997, verified: Foundational LSTM paper.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017, verified: arXiv:1706.03762 - foundational transformer architecture paper, NIPS 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[19] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020, verified: arXiv:2004.05150 - efficient long-document transformers with local attention. [Online]. Available: https://arxiv.org/abs/2004.05150

[20] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 283–17 297, 2020, verified: arXiv:2007.14062, NeurIPS 2020 - sparse attention for long sequences. [Online]. Available: https://arxiv.org/abs/2007.14062

[21] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," in *International Conference on Learning Representations*, 2021, verified: arXiv:2009.14794, ICLR 2021 - linear attention with FAVOR+ algorithm. [Online]. Available: https://arxiv.org/abs/2009.14794

[22] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," in *International Conference on Learning Representations*, 2022, verified: arXiv:2111.00396, ICLR 2022 - S4 state space sequence models. [Online]. Available: https://arxiv.org/abs/2111.00396

[23] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023, verified: arXiv:2312.00752 - Mamba selective state space models. [Online]. Available: https://arxiv.org/abs/2312.00752

[24] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances*

in Neural Information Processing Systems, vol. 28, 2015, verified: arXiv:1506.02626, NIPS 2015 - neural network pruning. [Online]. Available: https://arxiv.org/abs/1506.02626

[25] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019, verified: arXiv:1803.03635, ICLR 2019 - lottery ticket hypothesis. [Online]. Available: https://arxiv.org/abs/1803.03635

[26] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713, verified: arXiv:1712.05877, CVPR 2018 - quantization-aware training. [Online]. Available: https://arxiv.org/abs/1712.05877

[27] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015, verified: arXiv:1503.02531 - foundational knowledge distillation paper. [Online]. Available: https://arxiv.org/abs/1503.02531

[28] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *International Conference on Machine Learning*, pp. 6105–6114, 2019, verified: arXiv:1905.11946, ICML 2019 - efficient model scaling. [Online]. Available: https://arxiv.org/abs/1905.11946

[29] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021, verified: arXiv:1811.06128, DOI accessible - ML for combinatorial optimization survey. [Online]. Available: https://arxiv.org/abs/1811.06128

[30] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2019, verified: arXiv:1803.08475, ICLR 2019 - attention for TSP and VRP. [Online]. Available: https://arxiv.org/abs/1803.08475

[31] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 136–145, verified: arXiv:1703.00443, ICML 2017 - differentiable optimization layers. [Online]. Available: https://arxiv.org/abs/1703.00443

[32] A. Brandt, "Multi-level adaptive solutions to boundary-value problems," *Mathematics of Computation*, vol. 31, no. 138, pp. 333–390, 1977, verified: Foundational multigrid paper by Brandt.

[33] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid*. Academic Press, 2001, standard textbook on multigrid methods for solving PDEs.

[34] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," in *International Conference on Learning Representations*, 2021, verified: arXiv:2010.08895, ICLR 2021 - Fourier Neural Operator for PDEs. [Online]. Available: https://arxiv.org/abs/2010.08895

[35] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via deeponet based on the universal approximation theorem of operators," *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, 2021, verified: arXiv:1910.03193, DOI accessible - DeepONet operator learning. [Online]. Available: https://arxiv.org/abs/1910.03193