ML-Guided IBP Reduction

Neural Network Guided Integration-by-Parts Reduction of Feynman Integrals

Two-Loop Triangle-Box Topology

The Problem

- **IBP reduction**: Express complex Feynman integrals as linear combinations of simpler "master" integrals
- **Challenge**: Exponential search space of IBP identities
- **Traditional approaches**: Laporta algorithm, Kira, FIRE

The Memory Wall

Traditional IBP codes hit **memory limits** as integrals grow more complex:

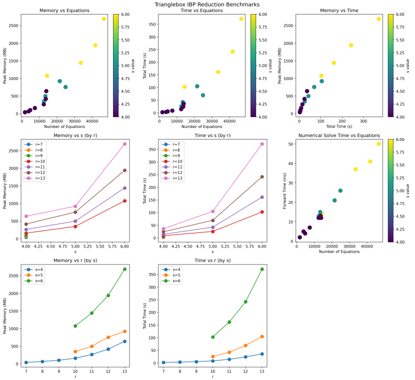

Figure 1: Kira Memory Scaling

Kira benchmarks: Memory grows exponentially with integral weight (r), reaching 2.5+ GB for r=13.

Our Solution

- **ML-guided beam search**: Train a neural network to score IBP actions
- **Hierarchical reduction**: Process sectors from highest to lowest
- **Parallel execution**: Distribute across Condor cluster for ~10x speedup
- **Constant memory**: Each one-step reduction is independent - no system accumulation

The Action Space: Key Innovation

Why Traditional IBP is Hard

- Infinite possible IBP identities (any seed, any operator)
- No clear way to choose which identity to apply
- Laporta: build giant linear system, solve globally

Our Approach: Finite Action Space

- Restrict to actions that **solve for a specific target integral**
- Each action = (IBP operator, seed shift)
- Enumerate only **valid actions** that eliminate the target
- Enables ML: classification over finite action set

Action Representation

Action = (ibp_op, delta)

- **ibp_op**: Which IBP/LI operator (0-15)
- **delta**: Shift from target to seed integral

Example

Target: `I[2,0,1,0,1,1,0]`, Action: `(3, (-1,0,0,0,0,0,0))`

```
IBP eq (op=3, seed=[1,0,1,0,1,1,0]):
  c₁·I[2,0,1,0,1,1,0] + c₂·I[1,0,1,...] + ... = 0
       ↑ target
```

Solve for target $\rightarrow$ express in terms of simpler integrals

Direct vs Indirect Actions

Direct Actions

- Target appears directly in the raw IBP equation
- Straightforward: apply IBP, solve for target

Indirect Actions (Key Innovation)

- Target does **not** appear in raw IBP equation
- But appears **after applying substitutions** from previous steps

```
Raw IBP:     c₁·I[A] + c₂·I[B] = 0   (no target)
After subs:  c₁·I[A] + c₂·(... + c₃·I[target] + ...) = 0
                                 ↑ target appears!
```

Indirect actions leverage reduction history for deeper reductions

Subsector Filtering

The Problem

Arbitrary IBP actions can introduce integrals in **higher sectors** $\rightarrow$ explosion

Solution: Subsector Constraint

Only allow actions where all resulting integrals are in **subsectors** of target

Sector Hierarchy

```
Sector 63 [1,1,1,1,1,1] (6 propagators)
    ↓ subsectors
Sector 62 [0,1,1,1,1,1] (5 propagators)
Sector 61 [1,0,1,1,1,1] (5 propagators)
    ↓
  ...lower sectors...
```

Result: Reductions flow downward through sector hierarchy

Why This Works

Finite + Learnable

- Typically 10-100 valid actions per state
- Model learns which actions lead to successful reductions

Hierarchical Structure

- Subsector filtering ensures monotonic progress
- Never introduces integrals harder than current target

Substitution Chains

- Indirect actions enable multi-step reasoning
- Model implicitly learns useful substitution patterns

Key Results

Triangle-Box Topology (arXiv:2502.05121)

Integral	Weight	Seq.	Par.	Speedup	Masters
`I[2,0,2,0,1,1,0]`	(6,0)	5 min	-	-	4
`I[1,1,1,1,1,1,-3]`	(6,3)	73 min	12 min	6x	16
`I[3,2,1,3,2,2,-6]`	(13,6)	$\sim$**20 hr**	**115 min**	$\sim$**10x**	16

- Results match Kira exactly
- Reduces to exact 16 paper masters from arXiv:2502.05121

System Architecture

```
┌─────────────────────────────────────────────┐
│              Training Pipeline              │
│  ┌─────────┐   ┌─────────┐   ┌─────────┐   │
│  │  Data   │ → │  Model  │ → │ Trained │   │
│  │Generation│   │ Training │   │Checkpoint│   │
│  └─────────┘   └─────────┘   └─────────┘   │
└─────────────────────────────────────────────┘
                       ↓
┌─────────────────────────────────────────────┐
│             Inference Pipeline              │
│  ┌─────────┐   ┌─────────┐   ┌─────────┐   │
│  │ Starting │ → │  Beam   │ → │ Master  │   │
│  │ Integral │   │ Search  │   │ Integrals│   │
│  └─────────┘   └─────────┘   └─────────┘   │
│                       ↓                     │
│       ┌─────────────────────────┐           │
│       │ Condor Parallelization  │           │
│       └─────────────────────────┘           │
└─────────────────────────────────────────────┘
```

Part 1: Data Generation

Data Generation: Scrambling Approach

Key Insight

Instead of collecting reduction trajectories (expensive), **reverse the process**: 1. Start from master integrals 2. Apply random IBP identities to increase complexity 3. Record each step - becomes training data when reversed

Constraints During Scrambling

- Only apply IBPs that don't introduce higher-sector integrals
- Stay within target sector and subsectors
- Ensures training data reflects valid reduction paths

Data Generation: Coverage

Sector Coverage

- All 63 non-trivial sectors covered
- Uses 16 paper masters for their respective sectors
- Uses corner integrals for remaining sectors

Dataset Statistics

Split	Samples	Size
Train	946,168	3.8 GB
Validation	118,271	480 MB
Test	~118,000	480 MB

Data Format

Each training sample contains:

```
{
    'sector_mask': [1,0,1,0,1,1],  # 6-bit sector encoding
    'expr': [                      # Current expression
        ([1,0,2,0,1,1,0], 107),    # (integral, coefficient)
        ([1,0,1,0,1,1,0], 303),
        ...
    ],
    'subs': [                      # Substitution history
        (key_integral, [(repl_int1, coeff1), ...]),
        ...
    ],
    'target_integral': [1,0,2,0,1,1,0],  # Integral to eliminate
    'valid_actions': [(ibp_op, delta), ...],
    'label': 3  # Index of correct action
}
```

Part 2: Model Architecture

Model: IBPActionClassifierV5

Overview

- **Task**: Score candidate IBP actions given current reduction state
- **Architecture**: Transformer-based with specialized encoders
- **Parameters**: 7.7M
- **Best validation accuracy**: 90.77%

Key Innovation (V5)

Full substitution encoding - encode not just the key integral but the complete replacement expression

Model Architecture Diagram

```
Inputs: Expression, Target, Substitutions, Sector, Actions
                          │
     ┌────────────────────┴────────────────────┐
     ▼                                          ▼
┌─────────────────────┐            ┌─────────────────────┐
│ Expression Encoder  │            │Substitution Encoder │
│ 2-layer Transformer │            │ 2-layer Transformer │
└──────────┬──────────┘            └──────────┬──────────┘
           │                                   │
           │     ┌─────────────────┐           │
           │     │ Sector Encoder  │           │
           │     └────────┬────────┘           │
           └──────────────┼────────────────────┘
                          ▼
                 ┌─────────────────────┐
                 │    State Combine    │
                 │   (4×256 → 256)     │
                 └──────────┬──────────┘
                          ▼
                 ┌─────────────────────┐
                 │   Cross-Attention   │
                 │ Actions → Expr Terms│
                 └──────────┬──────────┘
                          ▼
                    Action Scores
```

Component Details

Component	Purpose
Expression Encoder	Encode expression + target
Substitution Encoder	Encode reduction history
Sector Encoder	Condition on sector
Cross-Attention Scorer	Score actions vs expression

All use 2-layer architectures (Transformer or cross-attention)

Model Hyperparameters

- Embedding dimension: 256, Attention heads: 4
- Total parameters: 7,696,709

Part 3: Training

Training Configuration

```
epochs = 30
batch_size = 256
learning_rate = 0.0004
weight_decay = 1e-5
optimizer = AdamW
```

Training Results

- **Best checkpoint**: Epoch 22
- **Validation accuracy**: 90.77% (top-1)
- **Top-5 accuracy**: ~98%
- **Training time**: ~800s per epoch on GPU

Training Curves

Key Observations

- Model converges by epoch 20-25
- Validation accuracy plateaus at ~91%
- No significant overfitting observed
- Top-5 accuracy very high (~98%) - beam search can recover from top-1 mistakes

Part 4: Beam Search Inference

Beam Search Algorithm

Each Step (beam width = 20)

1. Start with **20 states** in the beam
2. For each state, identify **target** = max weight non-master
3. Model ranks valid actions; expand **top 20 actions per state**
4. This produces ~**400 candidate states**
5. Sort by (max_weight, n_non_masters, -model_score)
6. Keep **top 20** → new beam

Termination

- Stop when best state contains **only master integrals**
- Model guides *which actions to try*; weight reduction decides *what to keep*

Beam Search Optimizations

P1: Equation Caching (~3-10x speedup)

- IBP equation generation is expensive (sympy operations)
- Cache `get_raw_equation` results
- Reuse across beam states with shared history

P2: Batched Inference (~50x speedup)

- Prepare all action candidates as numpy arrays
- Single batched forward pass through model
- Eliminates per-action inference overhead

Combined Effect

- Per-step time: 10-90s $\rightarrow$ 0.1-5s
- Makes deep reductions feasible

Hierarchical Reduction Strategy

Algorithm

1. Find highest-level sector with non-master integrals
2. Run beam search to eliminate all non-masters in that sector
3. Move to next highest sector
4. Repeat until only masters remain

Example: I[2,0,2,0,1,1,0]

Sector	Level	Steps
53	4	53
52	3	17
49	3	4
37	3	42
21	3	46
...	...	...
Total	-	**176**

Beam Restart Strategy (V11+)

Problem

After weight improvement, beam often contains suboptimal states that will never succeed

Solution: Beam Restart

1. Run beam search until weight improves
2. **Stop and restart** with only the best state
3. Prunes dead ends, enables deeper exploration

Impact

- Essential for high-weight integrals
- `I[1,1,1,1,1,1,-3]`: 1,416 steps across 45 sectors
- `I[3,2,1,3,2,2,-6]`: 46,345 steps across 62 sectors

Part 5: Parallelization

Parallelization Motivation

The Bottleneck

- Sequential reduction of `I[3,2,1,3,2,2,-6]` takes ~20 hours
- Single-threaded beam search on CPU
- Many independent integrals could be processed in parallel

Key Insight

Each one-step reduction is independent - distribute across workers!

Async Parallel Architecture

```
┌───────────────────────────────────────────────┐
│               Main Controller                  │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐     │
│  │Work Queue│  │  Cache   │  │Straggler │     │
│  │(pending) │  │(memoize) │  │Detection │     │
│  └──────────┘  └──────────┘  └──────────┘     │
└───────────────────────────────────────────────┘
                      │
        ┌─────────────┼──────────────┐
        ▼             ▼              ▼
   ┌─────────┐   ┌─────────┐   ┌─────────┐
   │Worker 1 │   │Worker 2 │   │Worker N │
   │ (1 CPU) │   │ (1 CPU) │   │ (8 CPU) │
   └─────────┘   └─────────┘   └─────────┘
        │             │              │
        └─────────────┼──────────────┘
                      ▼
              Results (pickle)
```

Key Parallelization Features

1. Async Work Distribution
   - Submit all pending non-masters immediately
   - Don't wait for level synchronization
   - Process results as they arrive
2. Memoization Cache
   - Store: `integral → reduced expression`
   - Avoid redundant work (~55,000 cache hits!)
   - Critical when stragglers produce already-cached results
3. Straggler Detection
   - Jobs >30 min are killed and resubmitted
   - Resubmit with 8 CPUs (parallel beam search)
   - Prevents slow integrals from blocking progress

Parallel Performance
I[3,2,1,3,2,2,-6] Results

Metric	Value
Time (sequential)	~20 hours
Time (parallel)	**115 minutes**
Speedup	~**10x**
Jobs submitted	21,096
Stragglers resubmitted	24
Cache hits	55,075
Final masters	16

Part 6: Results

Validation Against Kira

I[2,0,2,0,1,1,0] (Sector 53)

Method	Masters	Result
Kira	4	Reference
Our V5	4	Matches exactly

Our reduction produces the **exact same master basis** as professional IBP software.

Scaling Results

Integral	Weight	Time	Steps	Masters
`I[2,0,2,0,1,1,0]`	(6,0)	5 min	176	4
`I[1,1,1,1,1,1,-3]`	(6,3)	12 min*	1,416	16
`I[3,2,1,3,2,2,-6]`	(13,6)	**115 min***	131,769	16

*With parallel execution

Final Reduction Example

I[3,2,1,3,2,2,-6] → 16 Paper Masters

```
Final expression (mod 1009):
  171 * I[1,0,1,1,1,1,0]
  854 * I[1,1,0,1,1,1,0]
  377 * I[1,1,1,1,1,0,0]
  160 * I[-1,1,1,1,1,0,0]
  100 * I[0,1,1,1,1,0,0]
  647 * I[1,-1,1,0,1,1,0]
    9 * I[1,-1,1,1,1,0,0]
  ... (16 masters total)
```

Matches arXiv:2502.05121 basis exactly!

Part 7: Conclusions

Key Contributions

1. **ML-guided IBP reduction** that matches professional software (Kira)
2. **Constant memory usage** - avoids the memory wall of traditional approaches
3. **Hierarchical beam search** with restart strategy for deep reductions
4. **Async parallel execution** with $\sim$10x speedup
5. **Straggler handling** for robust distributed computing
6. **Paper-masters-only mode** for clean minimal basis

Technical Innovations

Model

- Full substitution encoding (V5)
- Cross-attention action scoring
- Target-aware expression encoding

Inference

- Equation caching (3-10x speedup)
- Batched model inference (50x speedup)
- Beam restart strategy

Parallelization

- Async one-step distribution
- Memoization cache (~55k hits)
- Automatic straggler resubmission

Future Work

1. **GPU workers** for faster beam search
2. **Adaptive timeouts** based on sector statistics
3. **More integrals** - test on other two-loop families
4. **Training on successful paths** - use reduction results to improve model
5. **Path optimization** - shorten saved reduction paths

Code Availability

Repository: `github.com/davidshih17/RL_IBPreduction_claude`

Key Files

- `models/classifier_v5.py` - Model architecture
- `scripts/eval/hierarchical_reduction_async.py` - Parallel reduction
- `scripts/eval/reduce_integral_onestep_worker.py` - Condor worker
- `docs/parallelization.md` - Detailed documentation

Thank You

Questions?

Contact

- Repository:
  `github.com/davidshih17/RL_IBPreduction_claude`
- arXiv: 2502.05121 (paper masters reference)