ManuMatic: Strategy Injection for Robust Automatic Hybrid Parallelism in Distributed DNN Training

Ruiwen Wang[1,2,3], Chong Li[1], Hongxing Wang[1], Raja Appuswamy[3], and Yujie Yuan[1]

[1] Huawei Technologies France SASU, Boulogne-Billancourt, France
[2] Sorbonne University, Paris, France
[3] Eurecom, Biot, France
{wang.ruiwen,ch.l,yuan.yujie}@huawei.com,
raja.appuswamy@eurecom.fr

Abstract. Training modern deep neural networks (DNNs) requires hybrid parallelism. Automatic planners search data, tensor/model, and pipeline shardings with cost models, but decisions can drift from runtime optima due to framework/planner decoupling and overlap mis-modeling. We present ManuMatic, a light-touch planner that lets users *pin* a few critical operator shardings while automatically deriving globally consistent strategies for the rest. Inside a binary recursive partitioner, ManuMatic prioritizes pins via an infinite compromise price and decomposes multi-dimensional hints into two-way refinements; when hard constraints are infeasible, a soft-penalty variant applies. The design is profiling-free, preserves D-Rec's short compilation time, and degenerates to D-Rec when no pins are given. Built atop D-Rec, ManuMatic delivers consistent speedups without cost-model reengineering: on Mixtral-8×7B, an expert-parallel-aware BMM pin achieves 2.24× over D-Rec; on Llama3-8B, a sequence-parallel-aware MatMul pin reaches 2.04×; on Qwen2.5-72B, a sequence-parallel-aware MatMul pin combined with BMPipe yields 1.45× over D-Rec and 1.30× over an expert plan. These results show that minimal guidance can robustify automatic parallelism while largely preserving automation.

Keywords: Distributed training · Hybrid parallelism · Automatic planning · Strategy injection · Operator sharding · Cost model · Communication optimization · Deep neural networks · ManuMatic

1 Introduction

Deep learning has advanced rapidly over the last decade, powering major progress in computer vision and natural language processing. A persistent trend is the growth in architectural complexity and parameter counts of deep neural networks (DNNs) [1]–[3], which makes training on accelerator clusters the norm.

Large-scale training typically relies on *hybrid parallelism*, a composition of data, tensor/model, and pipeline parallelism, to satisfy memory constraints while sustaining device utilization [4]–[6]. Carefully engineered hybrids can be highly efficient; for instance, Megatron-LM [7] coordinates sharding and scheduling to unlock strong throughput on large transformers. However, hand-crafting such plans requires deep systems expertise, long iteration cycles, and offers no optimality guarantees due to the coupled, combinatorial design space.

Automatic planners [8]–[10] address this by searching over sharding and scheduling spaces guided by cost models. In practice, two gaps often degrade their decisions. (i) Decoupling between the framework and the planner. Training frameworks evolve (new kernels, fusions, collective algorithms, routing policies), while planner cost models do not automatically track these changes, so modeled and realized communication diverge. The mismatch is worse when the framework introduces new tensor split dimensions or new forms of parallelism (additional mesh axes, sequence [11]/expert parallel [12]–[14], optimizer/state partitioning). Without corresponding entries in the planner's cost model, dimension mapping $f(\cdot)$, and mismatch classifier, the search space is misspecified, especially for Mixture of experts and long-sequence DNN models. (ii) Overlap mis-modeling. The distributed computational graph used across frameworks does not encode runtime attributes such as compute/communication overlap, stream scheduling, or time-varying contention. Additive communication models become systematically optimistic in boundary regimes (tiny batches or skewed aspect ratios). Both gaps skew traversal priorities and push the search toward suboptimal shardings.

We present ManuMatic: a light-touch automatic planner that lets users *pin* the sharding of a *small* set of critical operators while automatically deriving globally consistent strategies for the rest of the graph. Within the binary recursive partitioner, ManuMatic assigns pinned operators an infinite *compromise price* (the minimal extra communication to avoid redistributions), ensuring they are fixed first, and decomposes multi-dimensional hints into per-level two-way refinements compatible with the recursion.

We instantiate ManuMatic on top of the state-of-the-art D-Rec planner [8]. On Mixtral-8×7B (8 NPUs), ManuMatic with a single expert-parallel-aware BMM pin yields a 2.24× speedup over D-Rec; on Llama3-8B (8 NPUs, L=8 K), a sequence-parallel-aware MatMul pin delivers 2.04×; on Qwen2.5-72B (64 NPUs, L=32 K), a sequence-parallel-aware MatMul pin combined with BMPipe [15] attains 1.45× over D-Rec and 1.30× over an expert plan.

Contributions.

- ManuMatic: a minimal, user-guided automatic planner that anchors a few critical operators and leaves the remainder to automation, improving robustness under framework–planner drift and overlap variability.
- A practical integration into a binary recursive partitioner via infinite compromise price and binary-compatible refinement of user hints, preserving global consistency without backtracking.

– A validated instantiation atop D-Rec with production-scale results on Mixtral-8×7B and Qwen2.5-72B, demonstrating substantial speedups without cost-model reengineering.

2 Background

2.1 Hybrid Parallelism in Distributed DNN Training

Large-scale training composes multiple parallelism modes to satisfy memory limits while sustaining device utilization. *Data parallelism* shards the global batch and synchronizes gradients across replicas [4]; *model/tensor parallelism* splits parameters and activations to reduce per-device memory pressure [5]; and pipelining with micro-batching overlaps forward/backward across stages [6], [7]. Handcrafted hybrids can be highly efficient yet demand deep expertise and provide no optimality guarantees due to the coupled, combinatorial design space.

2.2 Distributed Computational Graph and Strategy Representation

We model a training step as a data-flow graph $G = (V, E)$ used by frameworks such as MindSpore [16], TensorFlow [17], and PyTorch [18]. Vertices $V = \{o\}$ are operators; edges $E = \{\mathbf{T}\}$ are tensors. Each tensor $\mathbf{T}$ is a D-dimensional array with element type, shape $\text{shape}(\mathbf{T}) \in \mathbb{N}^D$, and a per-dimension distribution strategy $\text{strat}(\mathbf{T}) \in \mathbb{Q}^D$, where

$$\text{partshape}(\mathbf{T}) \;=\; \text{shape}(\mathbf{T}) \;\odot\; \text{strat}(\mathbf{T})$$

denotes the partition shape. We interpret $\text{strat}(\mathbf{T})[d] = \frac{1}{k_d}$ as sharding dimension d into k_d equal parts and $\text{strat}(\mathbf{T})[d] = 1$ as replication. For implementation, it is convenient to use the integer split vector $\text{split}(\mathbf{T}) \in \mathbb{N}^D$ with $\text{split}(\mathbf{T})[d] = k_d = \frac{1}{\text{strat}(\mathbf{T})[d]}$. Feasibility requires $\text{shape}(\mathbf{T})[d]$ divisible by k_d for all d.

Device mesh. Let the device mesh be an r-D grid $M \in \mathbb{N}^r$ with $\prod_{a=1}^{r} M[a] = d$ devices. A mapping $f : \{1, \ldots, D\} \to \{1, \ldots, r\} \cup \{\perp\}$ assigns each tensor dimension either to a mesh axis (sharding) or to $\perp$ (replication). A strategy is *mesh-consistent* iff $\prod_{d:f(d)=a} \text{split}(\mathbf{T})[d] \leq M[a]$ for all a, and the product over a equals the intended parallel degree.

While this graph abstraction is ubiquitous across modern DL frameworks, it does not encode runtime attributes, such as compute/communication overlap, collective algorithm choices, stream scheduling, or time-varying network contention; thus it cannot capture the overlap and communication variability that dominate end-to-end performance.

2.3 Communication-Centric Cost Model

On large language model training with even partitioning of regular tensors, end-to-end step time is primarily sensitive to *cross-device communication*. Under

weak or strong scaling, per-device compute stays flat or even decreases, while collective costs grow with participants (e.g., $\Theta((P-1)\alpha + \frac{P-1}{P}\beta n)$ or $\Theta(\alpha \log P + \beta n)$). Most sharding choices keep total FLOPs nearly unchanged but alter which collectives run and where redistributions appear, so they mainly perturb communication rather than arithmetic work. Practical schedules overlap compute with communication, which can mask kernel-time variations, whereas network latencies on critical paths remain exposed. Memory/divisibility and mesh capacity are enforced as feasibility constraints. Hence we minimize communication cost subject to feasibility; compute terms can be added if needed but are second-order in our regimes.

Collective costs. We use the standard α–β model: sending B bytes costs $\alpha + \beta B$; collective costs scale with algorithmic factors. For example, a ring all-reduce of an n-byte buffer over p devices costs

$$T^{\text{ring}}_{\text{allreduce}}(n,p) \;\approx\; 2(p-1)\alpha \;+\; 2\frac{p-1}{p}\,\beta n,$$

while a tree-based reduce-scatter/all-gather yields $\mathcal{O}(\alpha \log p + \beta n)$; our formulation abstracts these into a primitive Collective$(\cdot)$.

Redistribution across edges. Let edge $e = (u \to v)$ carry tensor $\mathbf{T}$ with producer-side strategy strat_u and consumer-side strategy strat_v. A *redistribution* occurs if $\text{strat}_u \neq \text{strat}_v$ in a way that cannot be satisfied by local reindexing/broadcast. We define a *mismatch classifier*:

$$\text{class}(\text{strat}_u, \text{strat}_v) \in \{\textsc{None}, \textsc{AllGather}, \textsc{ReduceScatter}, \textsc{AllToAll}\},$$

which picks the cheapest collective needed to reconcile the two layouts (e.g., sharded→replicated on the same dimension ⇒ AllGather; replicated→sharded ⇒ ReduceScatter; sharded on different dimensions ⇒ AllToAll). The redistribution cost on e is

$$C_{\text{redist}}(e; \text{strat}_u, \text{strat}_v) \;=\; \text{Collective}\big(\text{class}(\text{strat}_u, \text{strat}_v),\; \text{bytes}(\mathbf{T}),\; p_e\big),$$

with p_e the effective group size on that edge.

Operator-local costs. Some operator/strategy pairs require intrinsic collectives, e.g., data-parallel gradients for weights, tensor-parallel partial sums, or pipeline stage boundaries. We write

$$C_{\text{local}}(o; \text{strat}_{\text{in}}, \text{strat}_{\text{out}}) \;=\; \sum_{c \in \mathcal{C}(o, \text{strat}_{\text{in}}, \text{strat}_{\text{out}})} \text{Collective}(c),$$

where $\mathcal{C}$ enumerates collectives implied by the operator semantics under the chosen sharding.

Objective. Given a strategy assignment S over all tensors, the communication objective is

$$C_{\text{comm}}(S) \;=\; \sum_{e \in E} C_{\text{redist}}(e) \;+\; \sum_{o \in V} C_{\text{local}}(o).$$

Many planners minimize C_{comm} subject to mesh consistency and divisibility.

2.4 Redistribution-Aware Prioritization: Compromise Price

When deciding an operator with some neighbor strategies already fixed, it is useful to quantify how costly it would be to *change* its locally optimal strategy in order to *avoid* redistributions with those neighbors.

Let $\mathcal{S}_o$ be the feasible strategy set for operator o, and let $N^{\text{fix}}(o)$ be the set of adjacent operators already fixed. Define

$$C_o(s) \;=\; C_{\text{local}}(o; s) \;+\; \sum_{e=(o,\cdot) \text{ or } (\cdot,o)} C_{\text{redist}}(e;\; s,\; S_{\text{fix}}),$$

where S_{fix} provides neighbor strategies on $N^{\text{fix}}(o)$. The *compromise price* is

$$\text{price}(o) \;=\; \min_{s \in \mathcal{S}_o^{\text{compat}}} C_o(s) \;-\; \min_{s \in \mathcal{S}_o} C_o(s),$$

with $\mathcal{S}_o^{\text{compat}}$ the subset eliminating redistributions against $N^{\text{fix}}(o)$ (if empty, set $\text{price}(o) = +\infty$). This nonnegative quantity induces a natural *traversal priority*: operators with larger $\text{price}(o)$ are more expensive to move and should be decided earlier to reduce backtracking and global cost.

2.5 Automatic Planning as Recursive Partitioning

A scalable approach used in OptCNN [10], Tofu [9], and D-Rec [8] casts planning as a *recursive partitioner* guided by C_{comm}:

1. Order. Compute $\text{price}(o)$ and obtain a traversal order by decreasing price.
2. Partition. Traverse vertices and (i) choose for each a candidate split (dimension and degree), (ii) assign operators/tensors to one of two groups to minimize $C_{\text{local}} + C_{\text{redist}}$ across the cut.
3. Recurse. Repeat on each group until the number of partitions equals the device count d. The outer loop runs $\lceil \log_2 d \rceil$ times since the number of partitions doubles each recursion; the inner loop touches $|V|$ vertices per level.

This framework accommodates different cost models and search heuristics while providing clear hooks for priorities and constraints.

2.6 Limits of State-of-the-art Automatic Planning

Most planners minimize a *model-based* communication objective:

$$C_{\mathrm{mdl}}(S) = \sum_{e \in E} C^{\mathrm{mdl}}_{\mathrm{redist}}(e; S) + \sum_{o \in V} C^{\mathrm{mdl}}_{\mathrm{local}}(o; S).$$

However, the *runtime* makespan depends on the framework's actual primitives and on compute/communication overlap, which the graph abstraction cannot encode:

$$C_{\mathrm{rt}}(S; \Theta, \sigma) \;\approx\; C_{\mathrm{mdl}}(S) + E_{\mathrm{prim}}(S; \Theta) + E_{\mathrm{ovr}}(S; \Theta, \sigma).$$

Here E_{prim} captures primitive drift (changes in collective algorithms, kernel fusions, routing, or α–β), and E_{ovr} captures overlap mis-modeling (schedule-dependent non-additivity due to stream concurrency and contention). For two strategies S_1, S_2, define $\Delta_{\mathrm{mdl}}{=}C_{\mathrm{mdl}}(S_2){-}C_{\mathrm{mdl}}(S_1)$ and $\Delta E{=}\big(E_{\mathrm{prim}}{+}E_{\mathrm{ovr}}\big)\big|_{S_2} - \big(E_{\mathrm{prim}}{+}E_{\mathrm{ovr}}\big)\big|_{S_1}$. Then $\Delta_{\mathrm{rt}} \approx \Delta_{\mathrm{mdl}}{+}\Delta E$. If $|\Delta E| \geq \Delta^{\min}_{\mathrm{mdl}}$ (the model margin), the model-optimal choice can flip at runtime.

New split dimensions or new parallelism. Beyond parameter drift, frameworks may expand the device mesh and sharding vocabulary itself. If the runtime moves from an r-axis mesh $M \in \mathbb{N}^r$ to $M' \in \mathbb{N}^{r'}$ with $r' > r$, or adds new parallelism modes (e.g., sequence/expert parallel, optimizer/state partitioning), the planner's mapping $f : \{1, \ldots, D\} \to \{1, \ldots, r\} \cup \{\perp\}$ cannot place splits along the new axes. The search is then restricted to a misspecified subset $\mathcal{S}^{\mathrm{mdl}} \subsetneq \mathcal{S}^{\mathrm{rt}}$, effectively projecting the runtime-optimal $S^\star {\in} \mathcal{S}^{\mathrm{rt}}$ to $\Pi(S^\star) {\in} \mathcal{S}^{\mathrm{mdl}}$ with a non-negligible gap $C_{\mathrm{rt}}(S^\star){-}C_{\mathrm{rt}}(\Pi(S^\star))$. In practice, the mismatch classifier may also lack the new collective variants, which further inflates the error and increases the chance of wrong choices.

Impact on traversal. Priority heuristics such as compromise price are computed from C_{mdl}. When drift or overlap alter effective costs, priorities can invert, causing extra redistributions and a worse final plan. The method in Sec. 3 anchors a few sensitive operators to the framework's actual behavior, shrinking the error where it matters most.

3 ManuMatic: Strategy Injection atop D-Rec Automatic Planner

We build ManuMatic on top of the state-of-the-art planner D-Rec [8] because it is a strong, *profiling-free*, communication-centric system with short compilation time. Its binary-recursive design scales cleanly: only $T = \lceil \log_2 d \rceil$ levels and each level visits $|\mathcal{V}|$ operators once, which avoids heavy ILPs, autotuners, and runtime sampling. The modular structure (cost model, operator couplings, and a redistribution-aware priority heuristic) exposes clean hooks for priorities and constraints, which ManuMatic reuses without altering the core search; importantly, ManuMatic degenerates exactly to D-Rec when no pins are provided,

ensuring fair, like-for-like comparisons. Formally, given a device mesh with d devices and a computational graph $G = (V, E)$, D-Rec minimizes

$$C_{\text{comm}}(S) \;=\; \sum_{e \in E} C_{\text{redist}}(e; S) \;+\; \sum_{o \in V} C_{\text{local}}(o; S),$$

subject to mesh consistency and divisibility (Sec. 2), and proceeds in $T = \lceil \log_2 d \rceil$ outer recursions, each doubling the number of partitions. ManuMatic adds a lightweight interface that pins the sharding of a few critical operators and equips the partitioner with mechanisms to treat these pins as first-class constraints, without changing D-Rec's recursion skeleton.
(1) Priority ordering. For each operator o, compute its *compromise price* price(o) (the extra communication needed to avoid redistributions against already fixed neighbors) and traverse operators in decreasing price(o) to reduce backtracking.
(2) Bipartition per recursion. Following the order, choose a candidate split (dimension and degree) and assign o and its incident tensors to one of two groups to minimize $C_{\text{local}} + C_{\text{redist}}$. This yields two subgraphs that preserve data dependencies.
(3) Strategy materialization and recursion. The group assignment induces per-tensor strategies consistent with the mesh mapping. Update fixed neighbors and recurse on each subgraph. The inner loop visits $|V|$ operators per level; the outer loop runs T levels.

3.1 Strategy Injection as Constraints

As summarized in Algorithm 1, ManuMatic extends D-Rec [8] by treating a few user-provided shardings as first-class constraints within the same recursion skeleton.

Feasible set. Users specify a small set of injected operators $\mathcal{O}_{\text{inj}}$ and, for any incident tensor $\mathbf{T}$, a target integer split vector $\text{split}^\star(\mathbf{T})$. ManuMatic optimizes the same objective as D-Rec but over the constrained set

$$\mathcal{S}_{\text{feas}} = \Big\{ S : \text{ mesh-consistent, divisible, and } \forall\, \mathbf{T} \sim \mathcal{O}_{\text{inj}},\ \text{split}(\mathbf{T}) \succeq \text{split}^\star(\mathbf{T}) \Big\},$$

where $\succeq$ denotes element-wise divisibility ($a \succeq b \iff \forall d,\ a[d]$ is a multiple of $b[d]$). Injected splits act as lower bounds that later recursions may further refine.

Binary-compatible decomposition. Because the outer loop is binary, a multi-way injected split must be expressed as a monotone sequence $\{\text{split}^{(t)}(\mathbf{T})\}_{t=0}^{T}$ with $\text{split}^{(0)}$=$\mathbf{1}$, $\text{split}^{(T)} \succeq \text{split}^\star$, and $\prod_d \text{split}^{(t+1)}[d] / \prod_d \text{split}^{(t)}[d] = 2$. We use a greedy factorization: at recursion t, place one factor 2 on a dimension that still needs capacity toward $\text{split}^\star$; if no 2-factor remains, place a provisional 2 on the mesh axis intended for a q>2 factor (to be completed in later rounds). When $\prod_d \text{split}^\star[d] \nmid 2^T$, we realize the nearest feasible $\tilde{K} = 2^T$ with $\text{split}^{(T)} \succeq \text{split}^\star$ (over-provision) or fall back to a soft penalty (below).

Algorithm 1 ManuMatic (concise): Strategy Injection atop D-Rec

Require: Device mesh M with d devices; graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; injected ops $\mathcal{O}_{\text{inj}}$; target splits $\text{split}^\star(X)$ for tensors incident to $\mathcal{O}_{\text{inj}}$; couplings Φ; penalty λ (set λ=0 for hard mode)

Ensure: Strategy assignment $S = \{\text{split}(X)\}_{X \in \mathcal{E}}$

$T \leftarrow \lceil \log_2 d \rceil$; initialize $\text{split}(X) \leftarrow \mathbf{1}$ for all tensors; `Parts`$\leftarrow \{\mathcal{G}\}$
for $t = 0$ to $T - 1$ **do**
  `NewParts`$\leftarrow \emptyset$
  for all subgraph $\mathcal{H} = (\mathcal{V}_H, \mathcal{E}_H)$ in `Parts` **do**
    (1) One-step refinement
    for all $X \in \mathcal{E}_H$ incident to some $op \in \mathcal{O}_{\text{inj}}$ **do**
      Multiply $\text{split}(X)$ by 2 on a needed dimension toward $\text{split}^\star(X)$ (if mesh/divisibility allow)
    end for
    Enforce equalities from Φ across coupled tensor dimensions
    (2) Priorities
    for all $op \in \mathcal{V}_H$ **do**
      $\text{price}(op) \leftarrow +\infty$ if $op \in \mathcal{O}_{\text{inj}}$ else CompromisePrice(op)
    end for
    $\pi \leftarrow$ operators in $\mathcal{V}_H$ sorted by decreasing price
    (3) Constrained bipartition
    $(\mathcal{H}_L, \mathcal{H}_R) \leftarrow \text{Bipartition}\big(\mathcal{H}, \pi, \Phi\big)$ by minimizing $C_{\text{local}} + C_{\text{redist}} + \lambda \sum_{X \sim \mathcal{O}_{\text{inj}}} D(\text{split}(X), \text{split}^\star(X))$ (penalty active only if $\lambda > 0$)
    (4) Materialize and recurse
    Emit per-tensor $\text{split}(\cdot)$ for $\mathcal{H}_L, \mathcal{H}_R$ and push them into `NewParts`
  end for
  `Parts`$\leftarrow$`NewParts`
end for
return $S = \{\text{split}(X)\}_{X \in \mathcal{E}}$

Semantic propagation. An injected choice on $o \in \mathcal{O}_{\text{inj}}$ constrains its incident tensors via a dimension-coupling relation Φ_o (e.g., elementwise alignment; MatMul $(i,k) \times (k,j)$ couples k across inputs and i, j across outputs). After each refinement, ManuMatic enforces equalities $\text{split}(\mathbf{T}_1)[d_1] = \text{split}(\mathbf{T}_2)[d_2]$ for all $(d_1, d_2) \in \Phi_o$ to avoid local redistributions.

Priority integration. To fix injected neighborhoods first, ManuMatic sets $\text{price}(o)$=$+\infty$ for all $o \in \mathcal{O}_{\text{inj}}$, placing them at the top of the traversal. During bipartition, decisions are restricted to those consistent with the current $\text{split}^{(t)}$ and with Φ_o.

Hard vs. soft injection. If hard constraints are infeasible on a given mesh, we minimize

$$C_{\text{comm}}(S) \; + \; \lambda \sum_{\mathbf{T} \sim \mathcal{O}_{\text{inj}}} D\big(\text{split}(\mathbf{T}), \text{split}^\star(\mathbf{T})\big),$$

over strategies that are only mesh-consistent and divisible; $D(\cdot,\cdot)$ measures deviation (e.g., $D(a,b) = \sum_d \log \frac{a[d]}{\gcd(a[d],b[d])}$) and $\lambda{>}0$ tunes adherence. Setting $\lambda{=}{+}\infty$ recovers hard injection.

One recursion in ManuMatic. At level $t{=}0,\ldots,T{-}1$: (i) advance injected tensors by one binary refinement step and propagate constraints via Φ; (ii) compute priorities with $+\infty$ for injected operators; (iii) run D-Rec's bipartition under these constraints; (iv) materialize strategies and recurse. When no injections are provided, ManuMatic degenerates exactly to D-Rec.

4 Experiments

We summarize our setup and methodology in three parts (hardware, software, and experimental design), and then present two studies (generality and compatibility).

Hardware setup. Experiments run on a Huawei Ascend-910 (A910) AI cluster. Each compute node contains 8× A910–64GB NPUs. Intra-node communication uses a mesh topology via HCCL[4], providing up to 392 GB/s aggregate bandwidth per NPU. Inter-node communication connects multiple A910 nodes in a RoCE-based ring with 25 GB/s unidirectional link bandwidth per interface.

Software setup. We integrate our strategy-injection planner ManuMatic into MindSpore[5], and execute deep learning models within it. Using the framework's profiling utilities, we collect per-run *throughput* (tokens/s). Unless otherwise stated, measurements exclude warm-up and average over steady-state steps; hyperparameters and kernel switches follow MindSpore defaults.

Experimental design. We compare (i) Megatron (without expert/sequence parallel) – a strong software baseline lacking the specific parallelism mode indicated per model; (ii) D-Rec (auto) – the automatic planner without user guidance; (iii) Expert-Fine-tuned – a hand-tuned plan; and (iv) ManuMatic – our planner with a minimal, targeted operator pin.

4.1 Generality: Mixtral-8×7B and Llama3-8B on 8 NPUs

We evaluate two single-node (8 NPU) workloads to assess generality.

Mixtral-8×7B. ManuMatic injects a BMM (batched matmul) strategy that explicitly includes the expert-parallel mesh dimension for MoE projections, steering the search away from a costly redistribution path. This yields a 2.24× speedup over D-Rec and approaches the expert plan.

[4] Huawei Collective Communication Library: `https://gitee.com/ascend/cann-hccl`.
[5] MindSpore AI computing framework: `https://gitee.com/mindspore/mindspore`.

Table 1: Generality on Mixtral 8×7B (8 NPUs). Speedups relative to D-Rec (auto).

Planner	Throughput (tokens/s)	Speedup vs. D-Rec
Megatron (w/o expert parallel)	2730	0.93×
D-Rec (auto)	2935	1.00×
Expert-Fine-tuned	7337	2.49×
ManuMatic (BMM injection)	6603	2.24×

Llama3-8B (L=8 K). ManuMatic pins a MatMul sharding that explicitly includes the sequence-parallel split dimension, aligning sequence-wise partitions across attention and MLP matrix multiplications to avoid long-sequence redistributions. This targeted pin delivers a 2.04× speedup over D-Rec and slightly surpasses the expert-tuned plan (1.98×), showing that minimal guidance can steer the search to high-quality shardings even for dense models.

Table 2: Generality on Llama3-8B (8 NPUs). Speedups relative to D-Rec (auto).

Planner	Throughput (tokens/s)	Speedup vs. D-Rec
Megatron (w/o expert parallel)	22730	0.98×
D-Rec (auto)	23194	1.00×
Expert-Fine-tuned	45939	1.98×
ManuMatic (MatMul injection)	47455	2.04×

4.2 Compatibility: Qwen2.5-72B on 64 NPUs with BMPipe

We train Qwen2.5-72B on eight nodes (N_{NPU}=64, sequence length L=32 K). ManuMatic injects a MatMul sharding that explicitly includes the sequence-parallel split dimension, and we combine it with BMPipe[15]'s layer-assignment and recomputation optimizations. As shown in Table 3, this configuration attains a 1.45× speedup over D-Rec and a +30.1% improvement over the Expert-Fine-tuned plan.

Table 3: Compatibility on Qwen2.5-72B (64 NPUs, L=32 K). Speedups relative to D-Rec (auto).

Planner	Throughput (tokens/s)	Speedup vs. D-Rec
Megatron (w/o sequence parallel)	147	0.95×
D-Rec (auto)	155	1.00×
Expert-Fine-tuned	173	1.11×
ManuMatic + BMPipe	225	1.45×

Discussion. On Mixtral, a single expert-parallel-aware BMM pin removes a redistribution hotspot in MoE projections, delivering a 2.24× speedup over D-Rec while remaining close to an expert plan. On Qwen2.5-72B, a sequence-parallel-aware MatMul pin combined with BMPipe reduces stage idling and exposes additional compute/communication overlap, yielding 1.45× over D-Rec and 1.30× over the expert plan at cluster scale.

5 Conclusion

We presented ManuMatic, a light-touch strategy injection mechanism that robustifies automatic planning for hybrid parallelism in distributed DNN training. Conceptually, ManuMatic treats a small set of user-provided shardings as constraints that anchor the search to the framework's actual runtime behavior. Algorithmically, it integrates with a recursive partitioner: injected multi-way shardings are decomposed into per-iteration binary refinements, operator dimension couplings are enforced to avoid local redistributions, and traversal is steered by assigning injected operators an infinite compromise price. We also formalized why purely automatic planners may drift from the runtime optimum, through primitive drift across framework updates and schedule-dependent overlap slack, clarifying where minimal guidance is most effective.

Experiments on production-scale models show that very little guidance goes a long way: on Mixtral-8×7B (8 NPUs), a single expert-parallel-aware BMM pin yields a 2.24× speedup over D-Rec; on Llama3-8B (8 NPUs, L=8 K), a sequence-parallel-aware MatMul pin delivers 2.04×; on Qwen2.5-72B (64 NPUs, L=32 K), a sequence-parallel-aware MatMul pin combined with BMPipe attains a 1.45× speedup over D-Rec and 1.30× over an expert plan.

Future Work. We will extend ManuMatic from single-operator pins to interaction-aware *multi-pin* sets for larger models (long-context, MoE), adding conflict detection, simple ordering, and feasibility guards to avoid hidden reshards. We will quantify scalability as pin counts grow (compile time, planner memory, throughput) with warm starts, subgraph caching, column-generation refinements, and pruning, and report break-even points and negative interactions. Baselines will be broadened beyond D-Rec and expert plans to Alpa, FlexFlow, and SPMD planners in a unified harness, reporting step time, peak memory, communication volume, and compilation latency. For interpretability, we will attribute gains to eliminated redistributions/collectives, improved overlap, and stage rebalance, provide critical-path and communication visualizations, and report uncertainty (95% bootstrap CIs, effect sizes). We will study robustness to suboptimal or mis-placed pins via random/perturbed/adversarial ablations and add a diagnostic mode (counterfactual re-plans, sensitivity metrics), defaulting to a soft-penalty variant when hard constraints are risky. Finally, we will port ManuMatic beyond Ascend/MindSpore to GPU clusters and mainstream stacks (PyTorch/DTensor, XLA/StableHLO) with platform-agnostic pin representations, HCCL→NCCL adapters, and interconnect-aware cost-model calibration (NVLink vs. PCIe/IB), and replicate all analyses there.

References

[1] T. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, 2020, pp. 1877–1901.

[2] A. Yang *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[3] A. Q. Jiang *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[4] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] J. Dean *et al.*, "Large scale distributed deep networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada, 2012, pp. 1223–1231.

[6] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[7] M. Shoeybi *et al.*, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[8] H. Wang *et al.*, "Efficient and systematic partitioning of large and deep neural networks for parallelization," in *European Conference on Parallel Processing*, Springer, 2021, pp. 201–216.

[9] M. Wang *et al.*, "Supporting very large models using automatic dataflow graph partitioning," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19, Association for Computing Machinery, 2019.

[10] Z. Jia *et al.*, "Exploring hidden dimensions in accelerating convolutional neural networks," in *International Conference on Machine Learning*, J. Dy *et al.*, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Oct. 2018, pp. 2274–2283.

[11] V. Korthikanti *et al.*, "Reducing activation recomputation in large transformer models," *arXiv preprint arXiv:2205.05198*, 2022.

[12] N. Shazeer *et al.*, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *International Conference on Learning Representations (ICLR)*, 2017.

[13] W. Fedus *et al.*, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparse training," *arXiv preprint arXiv:2101.03961*, 2021.

[14] S. Rajbhandari *et al.*, "Deepspeed-moe: Advancing mixture-of-experts inference and training," *arXiv preprint arXiv:2201.05596*, 2022.

[15] R. Wang *et al.*, "BMPipe: Bubble-Memory Co-optimization Strategy Planner for Very-large DNN Training," in *IEEE International Conference on Cluster Computing 2025*, 2025.

[16] Huawei, *Mindspore*, `https://www.mindspore.cn/`, 2022.

[17] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, USENIX Association, Nov. 2016, pp. 265–283.

[18] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.