H^2O: Holistic Hyper-Parameter Optimization for Large-Scale Deep Neural Network Training

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

[1]Sorbonne University, Paris, France,
[2] EURECOM, Biot, France,
[3]Huawei Technologies France SASU, Paris, France
Email: {wang.ruiwen, ch.l, yuanyujie}@huawei.com, raja.appuswamy@eurecom.fr

1 Introduction and Motivation

Deep neural networks (DNNs) have evolved from research curiosities with millions of parameters to production systems exceeding hundreds of billions of parameters. Recently released foundation models—*GPT*, *Llama* and *DeepSeek* have redefined state-of-the-art performance. Scaling rules predict further accuracy gains with even larger models, the bottleneck has shifted from model expressiveness to how efficiently accelerator clusters are used during training. Modern practitioners must jointly choose data-, tensor-, pipeline- and optimizer-sharding degrees, micro-batch size μ, layer–stage assignment and recomputation strategies. Exhaustive search is infeasible; therefore, current planners limit themselves to a subset of axes, often leading to suboptimal throughput and memory waste.

This work presents H^2O, a two-level hyper-parameter optimizer that explores the complete cross-axis design space while remaining lightweight enough for everyday use. Level 1 selects the most cost-effective combination of data (DP), tensor (TP), pipeline (PP) and optimizer (OP) parallelism degrees together with the micro-batch size, explicitly balancing predicted execution time against per-device memory footprint. Level 2 then refines this tuple by jointly determining the layer stage assignment and the set of layers to be recomputed, to minimize the overall pipeline time.

Training trillion-parameter networks is infeasible on a single accelerator and therefore requires combining multiple orthogonal forms of distributed execution. The community has proposed a sequence of systems that automate different subsets of these techniques. We situate five representative systems within a common taxonomy of parallelism and run-time optimizations, and contrast these capabilities with the unified framework presented in this paper (Table 1): *D-Rec* [1] pioneers operator-level tensor sharding that minimizes cross-device bandwidth, but it neither pipelines computation nor shards optimizer state. *DAPPLE* [2] extends data parallelism with a synchronous pipeline planner. *PipeDream* [3] automates layer–device assignment and overlaps forward/backward passes via the 1F1B scheduler; however, parameter versioning inflates memory and micro-batch tuning remains manual. *Alpa* [4] unifies DP, TP and PP through a two-level JAX/XLA search that frequently matches expert-designed plans, albeit at considerable compile–search cost and with limited framework portability. Finally,

AdaPipe [5] introduces dynamic programming to trade recomputation for memory while migrating layers to balance stage latency, but supports only Transformer-like architectures and omits tensor/optimizer sharding.

The proposed solution H^2O extends the design space in two key ways. First, it simultaneously explores all four parallelism modes (DP, TP, PP, OP), whereas existing systems cover at most three. Second, it jointly optimizes micro-batch size, stage assignment, and fine-grained recomputation under a single, analytic compute–communication–memory model. This holistic treatment enables near-optimal throughput on models exceeding 100B parameters while maintaining strict memory budgets.

Table 1: Supported dimensions of SOTAs solution

System	**Parallelism**				**Optimizations**		
	DP	TP	PP	OP	μ-batch	Stage assign.	Recompute
D-Rec [1]	✓	✓					
DAPPLE [2]	✓		✓			✓	
PipeDream [3]	✓		✓			✓	
Alpa [4]	✓	✓	✓			✓	
AdaPipe [5]			✓			✓	✓
This work	✓	✓	✓	✓	✓	✓	✓

2 System Design

Problem Definition For a fixed DNN and accelerator cluster we minimize total training time by choosing $\mathbf{H} = \underbrace{\{d,\, m,\, o,\, s,\, \mu\}}_{\text{level 1}} \cup \underbrace{\{\mathcal{R},\, \mathcal{A}\}}_{\text{level 2}}$, subject to memory and communication constraints.

– Level-1 (parallelism degrees): d data, m tensor, o optimizer, s pipeline stages, μ micro-batches.
– Level-2 (pipeline balance): $\mathcal{R}$ recomputed layers, $\mathcal{A}$ layer–stage assignment.

Level-1-search We define **input symbolic variables** as: $\mathcal{V} = \{(c_\ell, m_\ell)\}_{\ell=1}^{L} \cup \{M_{\text{dev}},\, \tau_{\text{link}},\, \alpha,\, B,\, \pi\}$, where c_ℓ, m_ℓ are FLOPs and memory per layer, M_{dev} device memory, τ_{link} link matrix, α compute/communication overlap, B fixed global batch size and π the pipeline schedule.

We define **Delta cost model** as: each degree $x \in \{d, m, o, s, \mu\}$ is characterized by $\boldsymbol{\phi}(x) = \big(\Delta p, \Delta a, \Delta u, \Delta\alpha, \Delta t_{\text{comm}}, \Delta t_{\text{comp}}\big)$, capturing its impact on parameter memory Δp, activation memory Δa, optimizer-state memory Δu, overlap $\Delta\alpha$, communication time Δt_{comm} and computation time Δt_{comp}.

We evaluate **the impact of level-1-search** strategy on the execution characteristics of one layer ℓ under one micro-batch on each device. We track six quantities per layer $\big(p_\ell,\ a_\ell,\ u_\ell,\ t_\ell^{\text{fwd}},\ t_\ell^{\text{bwd}},\ t_\ell^{\text{rc}}\big)$, representing parameter, activation, optimizer-state sizes and forward, backward, recompute times. Communication overlap differs across collectives, so we introduce pattern-specific ratios $\alpha_{\text{grad}},\ \alpha_{\text{tp}},\ \alpha_{\text{opt}} \in [0, 1]$. Inter-node bandwidths are encoded by the link matrix $\tau_{\text{link}} \in \mathbb{R}^{K\times K}$. For each pattern $c \in \{\text{grad}, \text{tp}, \text{opt}\}$ we derive an effective

scalar bandwidth $\widehat{\tau}_c = \mathrm{EffBW}\big(c, \tau_{\mathrm{link}}\big)$, which is substituted into the latency formulae below. Let the per-device micro-batch size after data-parallel splitting be $s(d) = \frac{B}{d}$. **Data parallelism degree** d**.** Gradient all-reduce on parameters p_ℓ incurs latency $\big(p_\ell/\widehat{\tau}_{\mathrm{grad}}(d)\big)(1-\alpha_{\mathrm{grad}})$. Activations scale linearly with $s(d)$, as $\Delta a_\ell(d) = s(d)\, a_\ell$. **Tensor parallelism degree** m**.** Parameter shards reduce per-device parameter memory to $\Delta p_\ell(m) = \frac{p_\ell}{m}$, while collective messages amount to $V_{\mathrm{tp}}(m) = \rho_{\mathrm{tp}}\, p_\ell$; the associated latency is $\big(V_{\mathrm{tp}}(m)/\widehat{\tau}_{\mathrm{tp}}(m)\big)(1-\alpha_{\mathrm{tp}})$. **Optimizer parallelism degree** o**.** Sharding reduces optimizer-state memory to$\Delta u_\ell(o) = \frac{u_\ell}{o}$, and synchronizing these shards costs $V_{\mathrm{opt}}(o) = \rho_{\mathrm{opt}}\, u_\ell$, giving latency $\big(V_{\mathrm{opt}}(o)/\widehat{\tau}_{\mathrm{opt}}(o)\big)(1-\alpha_{\mathrm{opt}})$.

Level-2-Search We estimate the **memory of each stage** in pipeline by given a schedule π, we define $M_j(\pi) = g_\pi\Big(\{p_\ell, \tilde{a}_\ell(\pi, \mathcal{R}, a_\ell), u_\ell\}_{\ell \in \mathcal{A}^{-1}(j)}\Big)$, where the function $g_\pi(\cdot)$ applies the live-activation rules dictated by schedule π. A configuration is feasible when $M_j(\pi) \le M_{\mathrm{dev}}\ \forall j$. For each stage j the schedule-specific operator $f_\pi(\cdot)$ aggregates its assigned layers' forward, backward and recomputation times, producing a stage's latency $T_j(\pi) = f_\pi\Big(\{t_\ell^{\mathrm{fwd}}, t_\ell^{\mathrm{bwd}}, t_\ell^{\mathrm{rc}}\}_{\ell \in \mathcal{A}^{-1}(j)}\Big)$. With μ micro-batches in the pipeline, **overall pipeline time** is approximated by $T_{\mathrm{pipe}}(\pi, \mu) \approx \mu \max_{1 \le j \le s} T_j(\pi)$, the overall duration equals the micro-batch count μ multiplied by the latency of the slowest stage under schedule π. During tuning, we choose $\mathcal{R}$ and $\mathcal{A}$ (and, at Level 1, d, m, o, s, μ) so that every $M_j(\pi) \le M_{\mathrm{dev}}$ while minimizing $T_{\mathrm{pipe}}(\pi, \mu)$.

We embed the two-level optimization in the pseudo code of Algorithm 1, which first searches for a best execution time/memory cost parallelism tuple $\big(d, m, o, s, \mu\big)$ and then refines the pipeline schedule by selecting $\mathcal{R}$ and $\mathcal{A}$.

Algorithm 1 Two-level search strategy

Require: symbolic variables $\mathcal{V}$, schedule π, search budget N
Ensure: best configuration $\mathbf{H}^\star$

$T_{\mathrm{best}} \leftarrow \infty$, $\mathbf{H}^\star \leftarrow \varnothing$
for $i = 1$ **to** N **do**
  $(d, m, o, s, \mu) \leftarrow$ Level-1-Search(ϕ)
  if MemExceeded$(d, m, o, s, \mu, \mathcal{V})$ **then continue**
  end if
  $(\mathcal{R}, \mathcal{A}) \leftarrow$ Level-2-Search(d, m, o, s, μ, π)
  $T \leftarrow T_{\mathrm{pipe}}(\pi, \mu)$
  if MemExceeded$(\mathcal{R}, \mathcal{A})$ **then continue**
  end if
  if $T < T_{\mathrm{best}}$ **then**
    $\mathbf{H}^\star \leftarrow \{d, m, o, s, \mu, \mathcal{R}, \mathcal{A}\}$; $T_{\mathrm{best}} \leftarrow T$
  end if
end forreturn $\mathbf{H}^\star$

Level-1-Search draws candidates by the Delta cost model ϕ. MemExceeded rejects tuples for which any stage violates the memory constraint $M_j(\pi) \le M_{\mathrm{dev}}$. Level-2-Search is a local search that adaptively balances recomputation strategies and layer-stage assignment to achieve the minimum overall pipeline time.

3 Experimental Evaluation

Experimental Setup Experiments are executed on 16x Huawei Atlas Ascend 910 computing nodes with CANN 7.2, MindSpore 2.5. Each node is equipped with 8x Ascend-910 NPUs, interconnected via a high-speed HCCS with a bandwidth of 392 GB/s per NPU. Multiple nodes are connected via a ring topology by RoCE network interfaces supporting a unidirectional bandwidth of 25 GB/s.

Baselines and Metrics We compare our *H²O* planner on a 141-billion-parameter *DeepSeek* model, against two widely adopted baselines: a) Baseline-A: algorithmic D-Rec. A state-of-the-art auto-parallel planner that searches data and tensor degree. It does not support optimization over pipeline or optimizer, nor does it explore layer assignment or recomputation. b) Baseline-B: expert hand-tuned. An expert uses a greedy search to sequentially tune data, tensor, pipeline, and optimizer degree, as well as micro-batch size and recomputation. To compare, we report (i) model-flop utilization (MFU) and (ii) iteration time.

Result As summarized in Table 2, H^2O achieves the best performance among all solutions. These results confirm that jointly optimizing parallelism degrees and pipeline balancing translates directly into higher accelerator throughput.

Table 2: Performance comparison on DeepSeek-141B with 128 Ascend NPUs.

Planner	Iter. Time (s) ↓	MFU (%) ↑	Speedup (%)
D-Rec	49.600	26.13	+0.00
Expert-tuned	44.396	29.34	+17.21
H^2O (Ours)	36.278	35.73	+36.72

4 Conclusion

We introduced *H²O*, a two-level hyper-parameter optimizer that searches data, tensor, pipeline and optimizer parallelism together with micro-batch size, stage assignment and adaptive recomputation. On a 128-NPU cluster training a 141-billion-parameter *DeepSeek* model, *H²O* gain a speedup up to 36.72%, outperforming both the state-of-the-art auto-parallel planner and expert-tuned.

References

[1] H. Wang *et al.*, "Efficient and Systematic Partitioning of Large and Deep Neural Networks for Parallelization," in *Lecture Notes in Computer Science*, ser. Lecture Notes in Computer Science, vol. 12820, Lisbon, Portugal: Springer International Publishing, 2021, pp. 201–216.

[2] S. Fan *et al.*, "Dapple: A pipelined data parallel approach for training large models," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '21, Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 431–445.

[3] D. Narayanan *et al.*, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19, Huntsville, Ontario, Canada: Association for Computing Machinery, 2019, pp. 1–15.

[4] L. Zheng *et al.*, "Alpa: Automating inter- and Intra-Operator parallelism for distributed deep learning," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, Carlsbad, CA: USENIX Association, Jul. 2022, pp. 559–578.

[5] Z. Sun *et al.*, "Adapipe: Optimizing pipeline parallelism with adaptive recomputation and partitioning," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24, La Jolla, CA, USA: Association for Computing Machinery, 2024, pp. 86–100.