Energy Efficient Inference and Training

Harsh, Shmeelok, Peter, Divyam

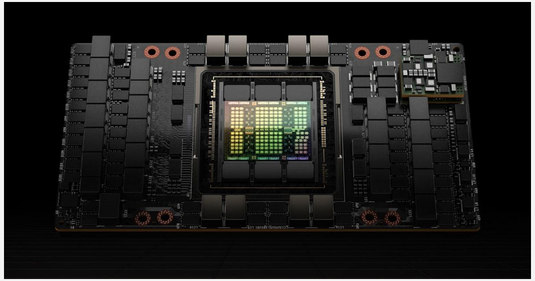
DynamoLLM:

Energy Management Framework for Inference Tasks

Jovan Stojkovic, Chaojie Zhang, Inigo Goiri, Josep Torrellas, Esha Choukse

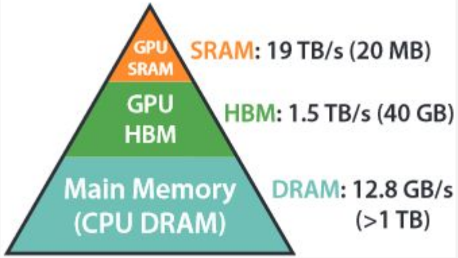
Motivation

Inference Tasks

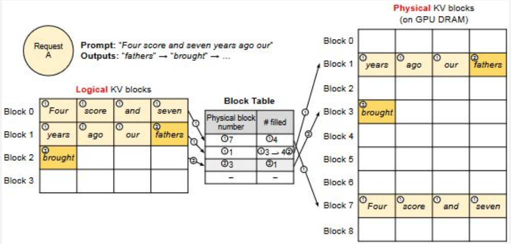


Existing Solutions

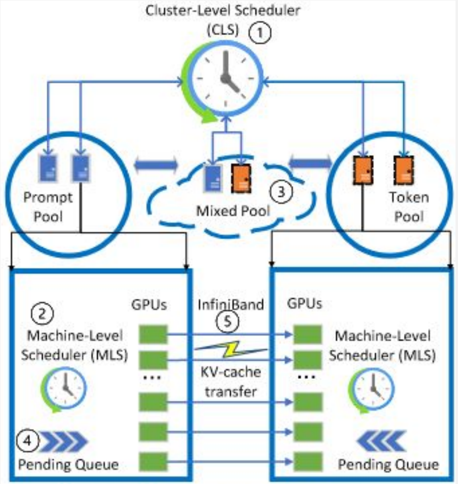
Flash Attention



Paged Attention

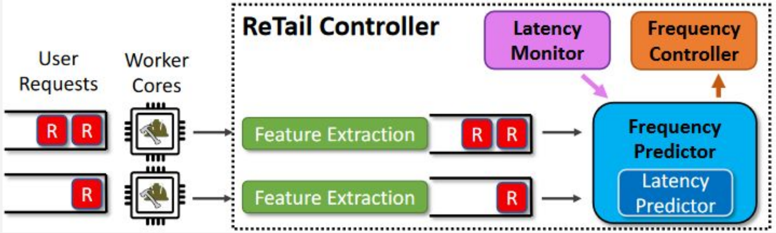


SplitWise

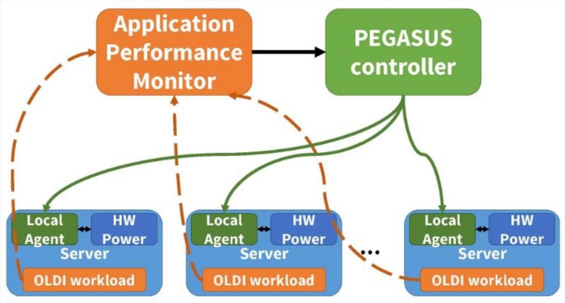


Relevant Work

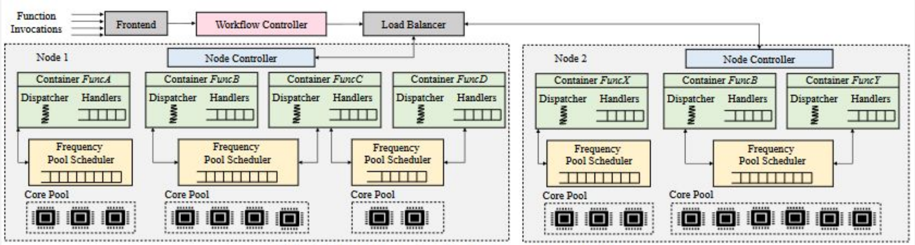
Retail



Pegasus



EcoFaaS



Insight #1

“There are opportunities for energy efficiency based on different input/output types”

Tensor Parallelism		TP2				TP4				TP8			
GPU Frequency (GHz)		0.8	1.2	1.6	2.0	0.8	1.2	1.6	2.0	0.8	1.2	1.6	2.0
Input	Output												
Short	Short		**0.77**	0.97	1.03	0.94	0.79	0.91	1.01	1.35	1.19	1.29	1.49
Short	Medium		**2.78**	3.45	3.68	3.39	2.82	3.37	3.81	4.55	4.15	4.43	4.74
Short	Long					4.84	**4.17**	4.97	5.52	6.37	5.62	5.59	6.95
Medium	Short			**1.02**	1.09		1.08	1.07	1.20	1.51	1.29	1.34	1.73
Medium	Medium						4.23	**3.91**	4.08	5.34	4.39	4.56	5.44
Medium	Long						4.99	4.66	**4.53**	6.86	5.79	6.52	7.12
Long	Short						**1.51**	1.64	1.76	2.55	2.53	2.83	2.94
Long	Medium										**7.71**	8.81	9.17
Long	Long										12.99	**11.89**	13.21

TABLE I: Energy consumption in Watt×hours (Wh) for Llama2-70B varying request lengths, frequency, and model parallelism with medium system load (2K tokens per second). Configurations that violate the SLO are shown as empty gray boxes, while the acceptable configurations are colored as a heat map according to their energy consumption, per row.

Insight #2

“LLM workloads are highly dynamic and the optimal configuration for energy savings changes constantly”

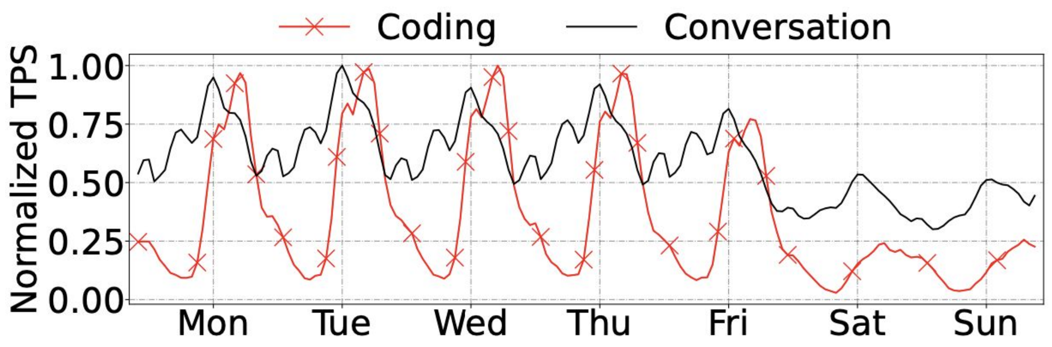


Fig. 2: Load over a week for *Coding* and *Conversation* LLM inference workloads.

Insight #3

“Changing LLM server configurations has significant overhead and must be understood/minimized.”

Overhead source	Time
Create a new H100 VM [36]	∼1-2 min
Initialize distributed multi-GPU environment	∼2 min
Download model weights (Llama2-70B [67])	∼3 min
Set up the engine configuration	∼18 sec
Install weights and KV cache on GPUs	∼15 sec
Total	∼6-8 min

TABLE V: Measured overheads of creating a new 8×H100 instance of an LLM inference server VM.

Key Design Requirement

“Achieving SLOs at any cost while optimizing Energy Consumption”

Energy/Performance Profile

Load

Request Length

Model Parallelism

GPU Frequency

$$f(x_0,\ldots,x_n)$$

Expected Energy Consumption

TTFT/TBT Latency

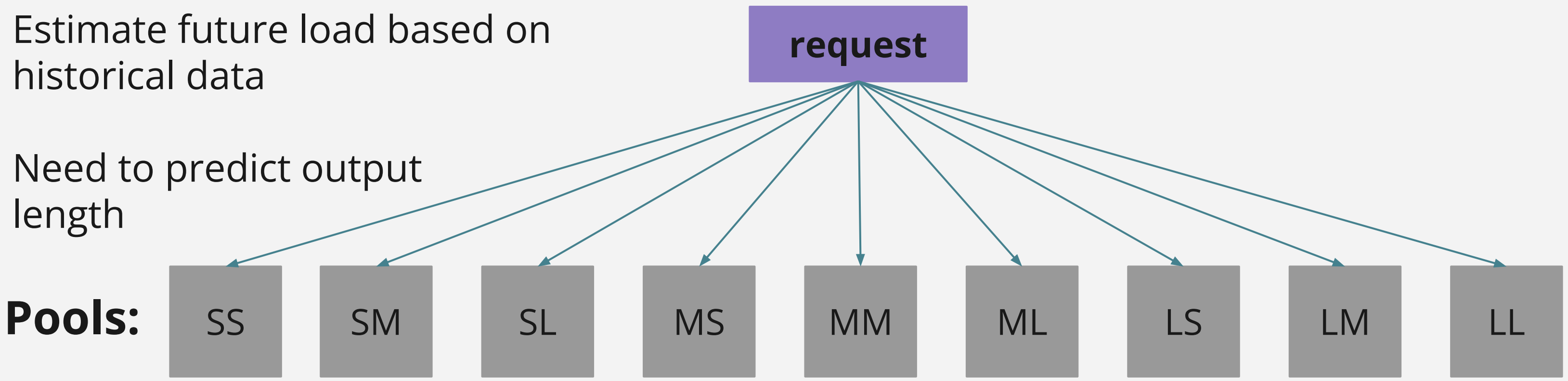
Pools

Insight #1: “There are opportunities for energy efficiency based on different input/output types.”

Solution: Allocate specific # of GPUs for each request type

Estimate future load based on historical data

Need to predict output length



Optimization Problem

Inputs:
- Energy/Performance Profile
- Current Load
- Hardware Available

Outputs:
- Model + Tensor Parallelism
- GPU Frequency

Constraints:
- SLOs met
- Hardware Constraints

$$
\begin{aligned}
\min \quad & \left(\sum_i (N^{TP_i} \times Energy^{TP_i,f_i}(L^{TP_i})) \right) \quad \forall i \in \{2,4,8\} \\
\text{s.t.} \quad & \sum_i i \times N^{TP_i} \leq N \\
& \sum_i (N^{TP_i} \times L^{TP_i}) \geq L \\
& Performance^{TP_i,f_i}(L^{TP_i}) \leq SLO
\end{aligned}
$$

HIGH COMPUTE COST

Pool Management

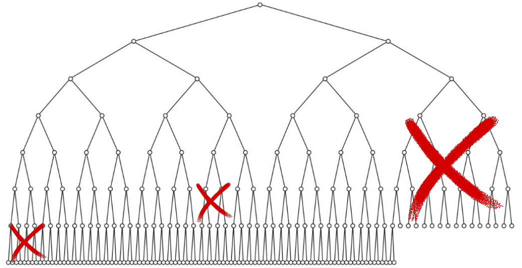
Goal: Determine Model + Tensor Parallelism

- Minimize energy consumption

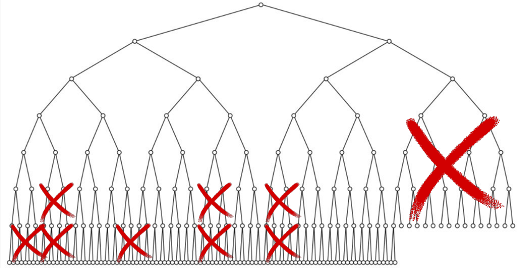
Search Space: Energy/Performance Profile

Idea: Prune non viable solutions while fixing other parameters

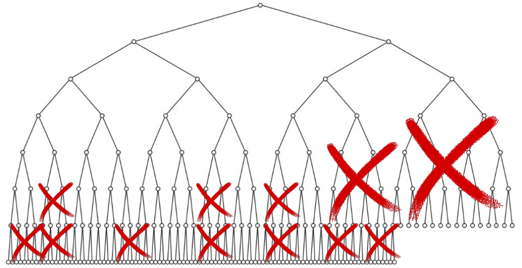
Available GPUs



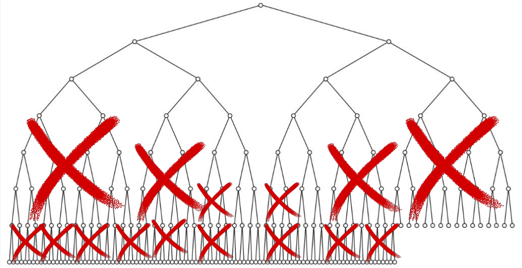
Predicted Peak Load



Max GPU Frequencies



Satisfies SLOs



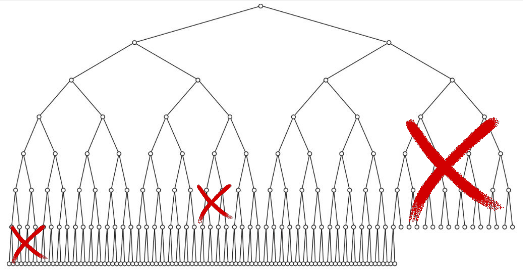
GPU Management

Goal: Determine GPU Frequency
- Minimize energy consumption

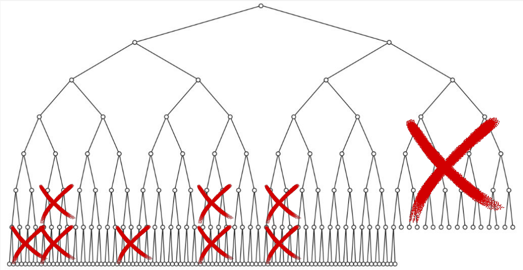
Search Space: Energy/Performance Profile

Idea: Prune non viable solutions while using computed parameters

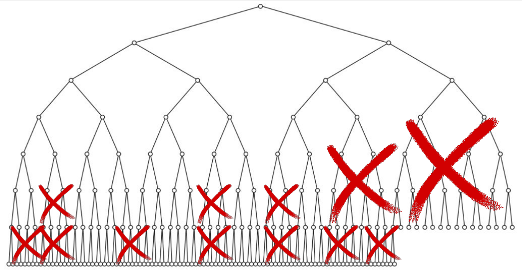
Current Load



Model + Tensor Parallelism



Satisfies SLOs



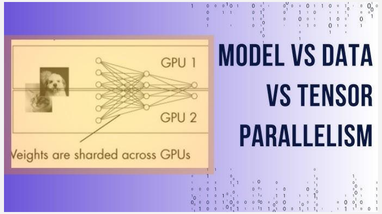
Advantages of Approximation

Insight #2: “LLM workloads are highly dynamic and the optimal configuration for energy savings changes constantly.”

- Approximate solution to optimization problem
- Change parameters with lower overhead at higher intervals

Lower Overhead

Higher Overhead



Reconfiguration Overheads

Insight #3: “Changing LLM server configurations has significant overhead and must be understood/minimized.”

- Maintain historical data of reconfiguration overhead
- Efficient algorithms to transfer model weights
- VM instantiation from snapshots

Expected Energy Consumption (current)

>

Reconfiguration Energy Consumption

+

Expected Energy Consumption (new)

Emergency Events

Output Length Misprediction Occurs

- Route request to next higher-performance pool

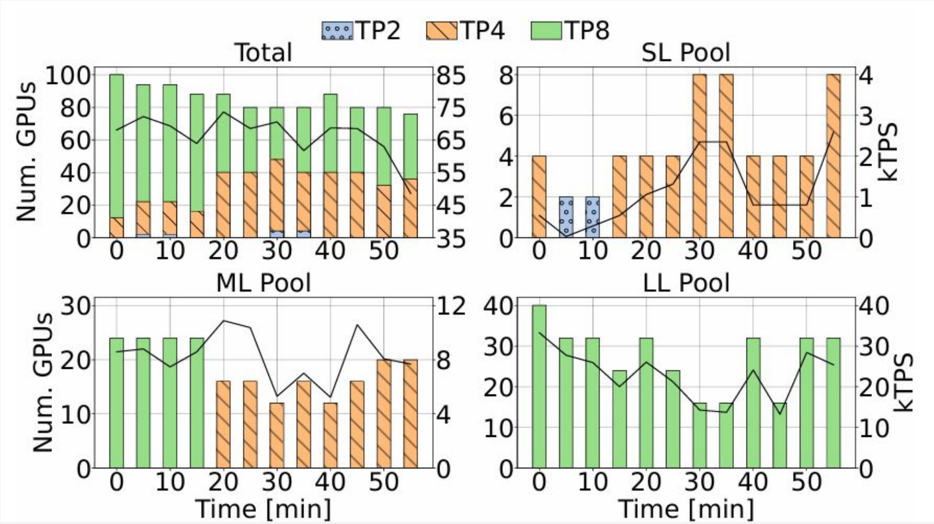
Excess queued requests

- Repriotize request based on closest to missing deadline
- Ramp up GPU frequency
- Route request to next higher-performance pool

Evaluation

Setup

- 8 H100 GPUs
- Llama2-70B
- *Coding* and *Conversation* Traces
- DynamoLLM + "state-of-the-practice" baseline systems



Perseus

Reducing Energy Bloat in Large Model Training

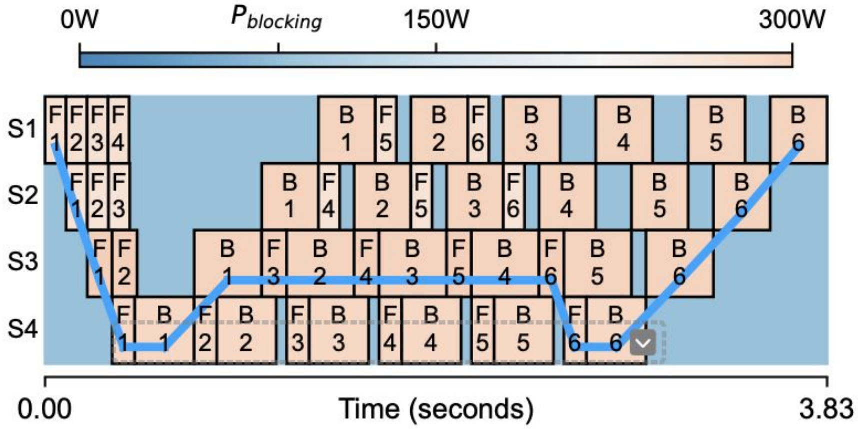
Jae-Won Chung, Yile Gu, Insu Jang, Luoxi Meng, Nikhil Bansal, Mosharaf Chowdhury

Motivation

- **Context:**
  - Amazon’s training of a **200B model**
  - Energy consumed: **11.9 GWh**
- **Observation:**
  - Not **all energy** consumed **during training** directly contributes to end-to-end throughput
  - Significant Energy can be removed **without slowing** down training, which the paper calls **ENERGY BLOAT**

Energy Bloat

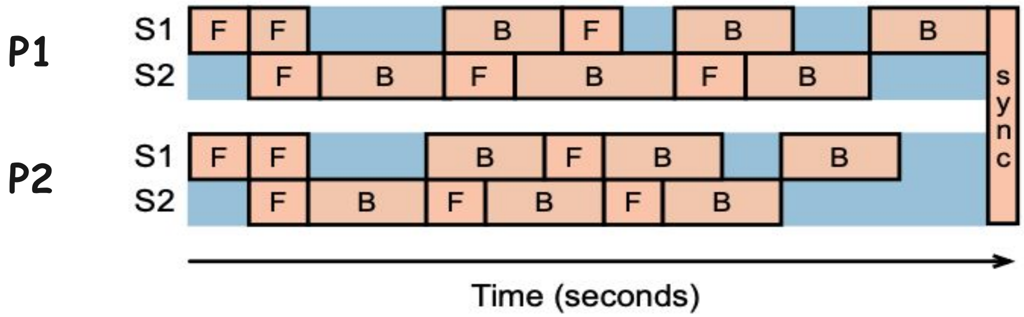
- **Origin of Energy Bloat:**
  - LLM Training is structured using 3D(Data, Tensor, Pipeline) parallelism
  - **Bottleneck 1:**
    - In **Pipeline Parallelism**, often GPUs are simply blocked on communication with an adjacent stage



(a) Execution timeline of one training iteration

Energy Bloat

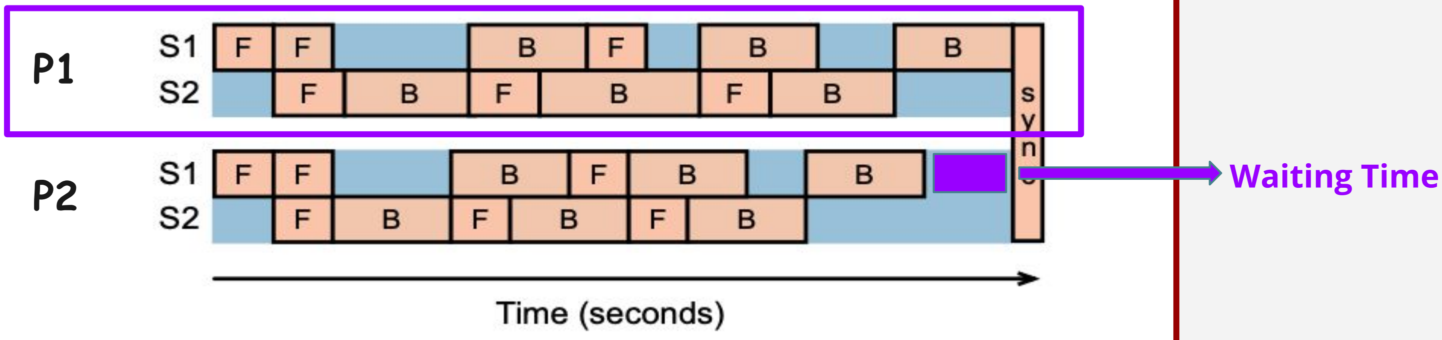
- **Origin of Energy Bloat:**
  - LLM Training is structured using 3D(Data, Tensor, Pipeline) parallelism
  - **Bottleneck 2:**
    - When these pipelines are replicated for **Data Parallelism**, **faster pipelines wait for the slower (straggler) one**



(a) Extrinsic energy bloat caused by a straggler

Energy Bloat

- **Origin of Energy Bloat:**
  - LLM Training is structured using 3D(Data, Tensor, Pipeline) parallelism
  - **Bottleneck 2:**
    - When these pipelines are replicated for **Data Parallelism**, **faster pipelines wait for the slower (straggler) one**

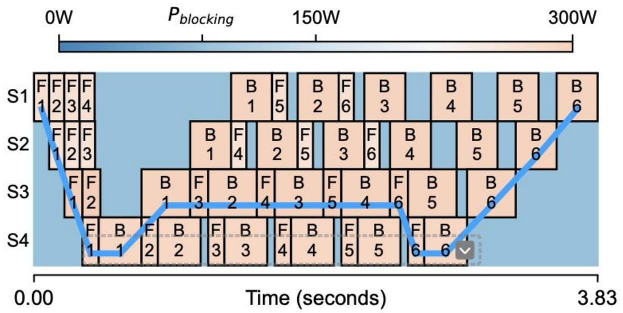


(a) Extrinsic energy bloat caused by a straggler

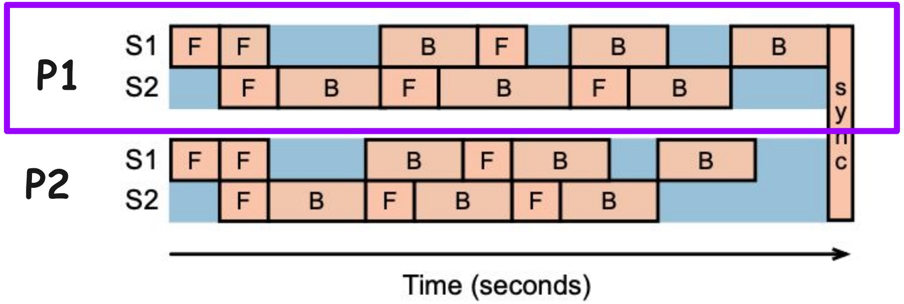
Energy Bloats

- **Intrinsic Energy Bloat:** Energy wasted due to computation imbalance across pipeline stages, causing **non-critical** computations to run **needlessly fast**

- **Extrinsic Energy Bloat**: Occurs when multiple pipelines run synchronously, and one pipeline (the **straggler**) **slows down**, forcing others to waste energy by waiting



(a) Execution timeline of one training iteration

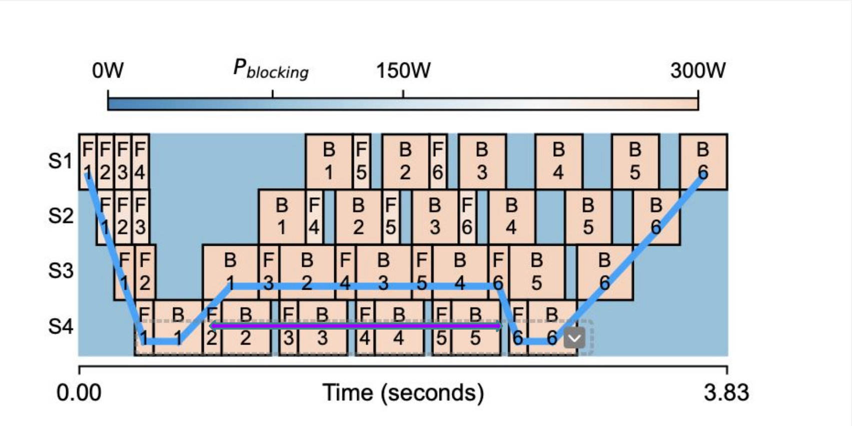


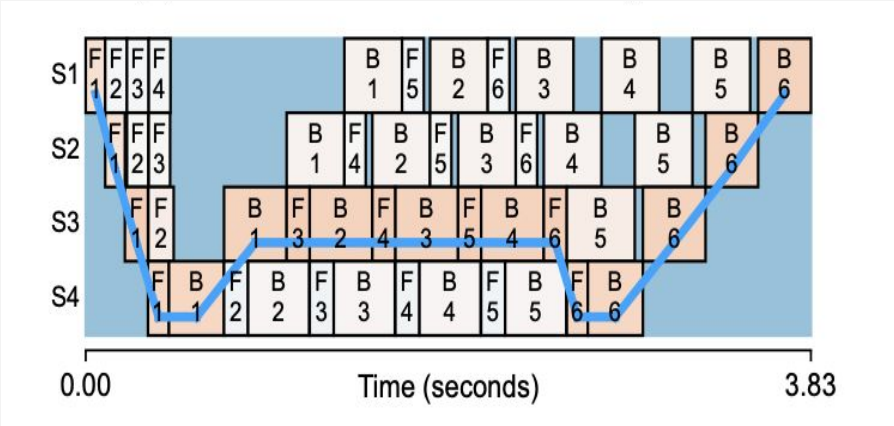
(a) Extrinsic energy bloat caused by a straggler

Unified Optimization Framework

Two Intuitions:

1. **Intuition 1: Slowing Non-Critical Computations for Intrinsic Energy Bloats Mitigation**

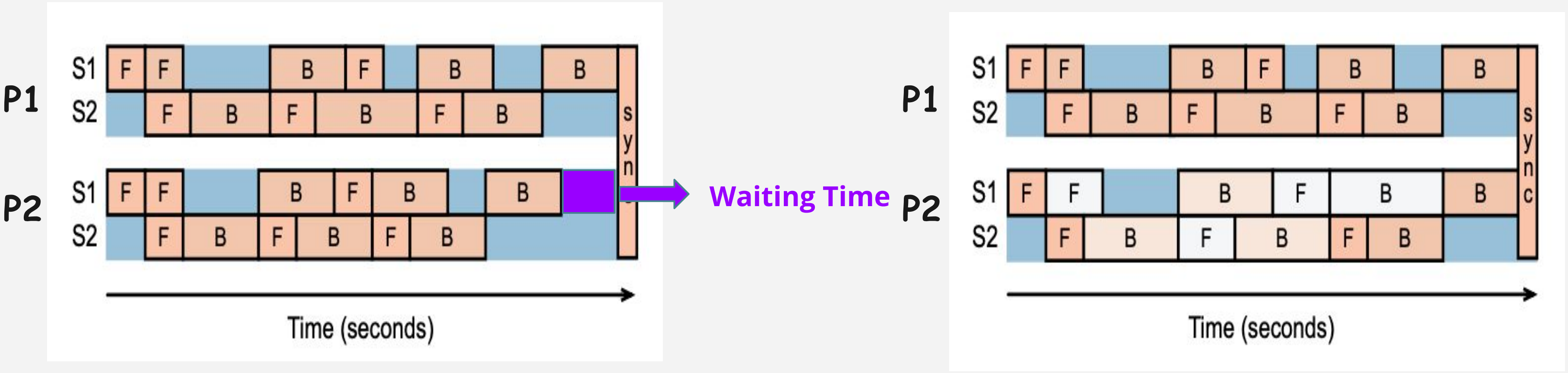




Unified Optimization Framework

Two Intuitions:

1. **Intuition 2: Adapting Non-Straggler Pipelines to Stragglers for Extrinsic Energy Bloats Mitigation**
    a. Prevents the faster pipelines from wasting energy while waiting for slower ones to complete



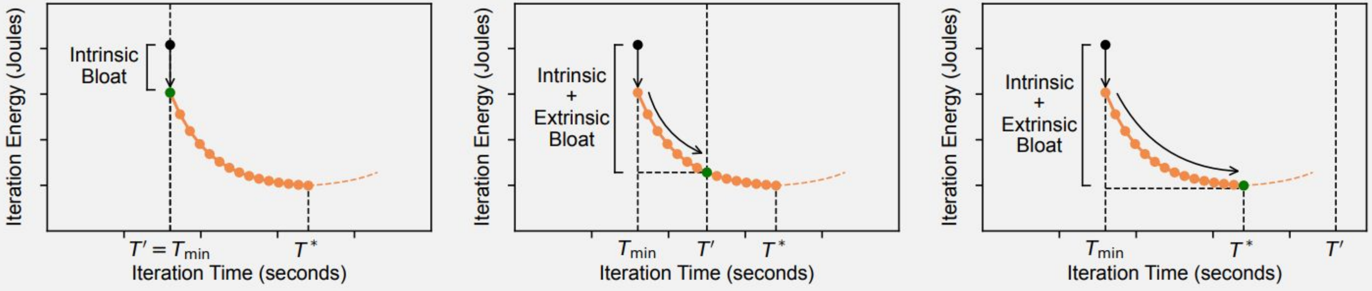
Frontier Time-Energy Characterization

Characterizing Goals

Goal: For a given straggler time T’, find the minimum energy an iteration could take given possible GPU frequencies

This is NP-Hard for all approximations!

Instead: Try a continuous approximation with continuous frequencies



(a) $T' = T_{min}$ **(b)** $T_{min} < T' \leq T^*$ **(c)** $T^* < T'$

Finding the Frontier

1. Start from **minimum possible energy**
2. Decrease the time **by some pre-determined time τ** and then **iteratively** get the schedule with minimal energy increase

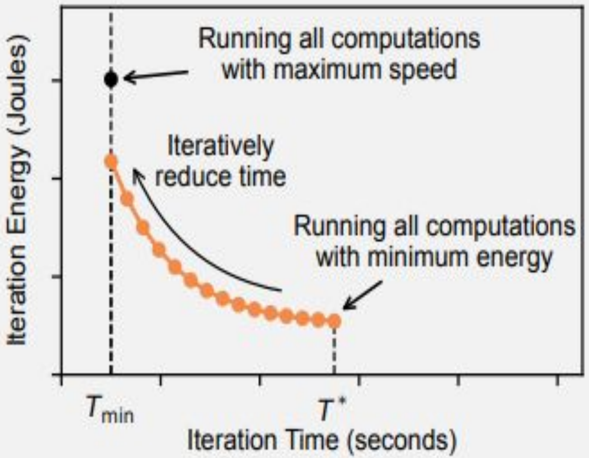
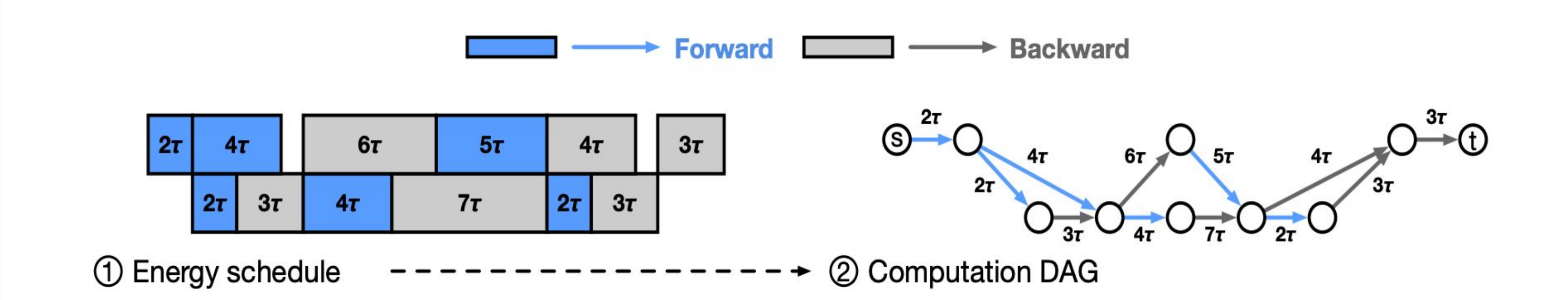


Figure 5. Starting from the energy schedule that consumes the minimum energy, we iteratively reduce its iteration time to trace up and iteratively discover the tradeoff frontier.

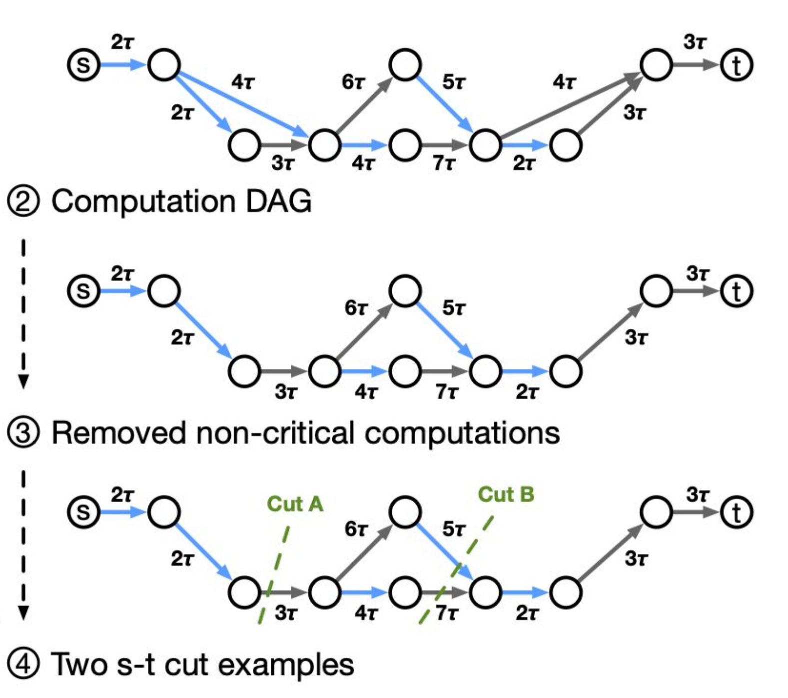
Neighbor Energy Schedule – DAG

- **Pipeline Representation**: Timeline with gaps due to dependency stalls
- **DAG Representation**: Show computation dependencies as nodes and computations as edges (each with a time + energy)



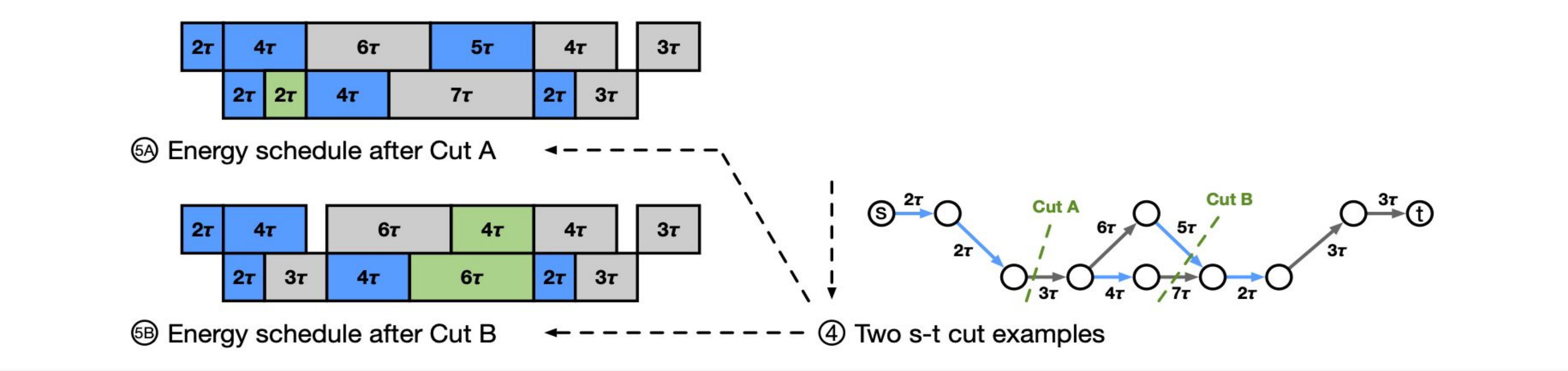
Neighbor Energy Schedule – Critical Paths and Min Cuts

- **Goal**: Reduce all critical paths by τ
- **Insight 1**: Non-critical paths don't need to be reduced
- **Insight 2**: An s-t cut represents a way to reduce all critical paths by τ
- **Conclusion**: Find the min-cost cut!

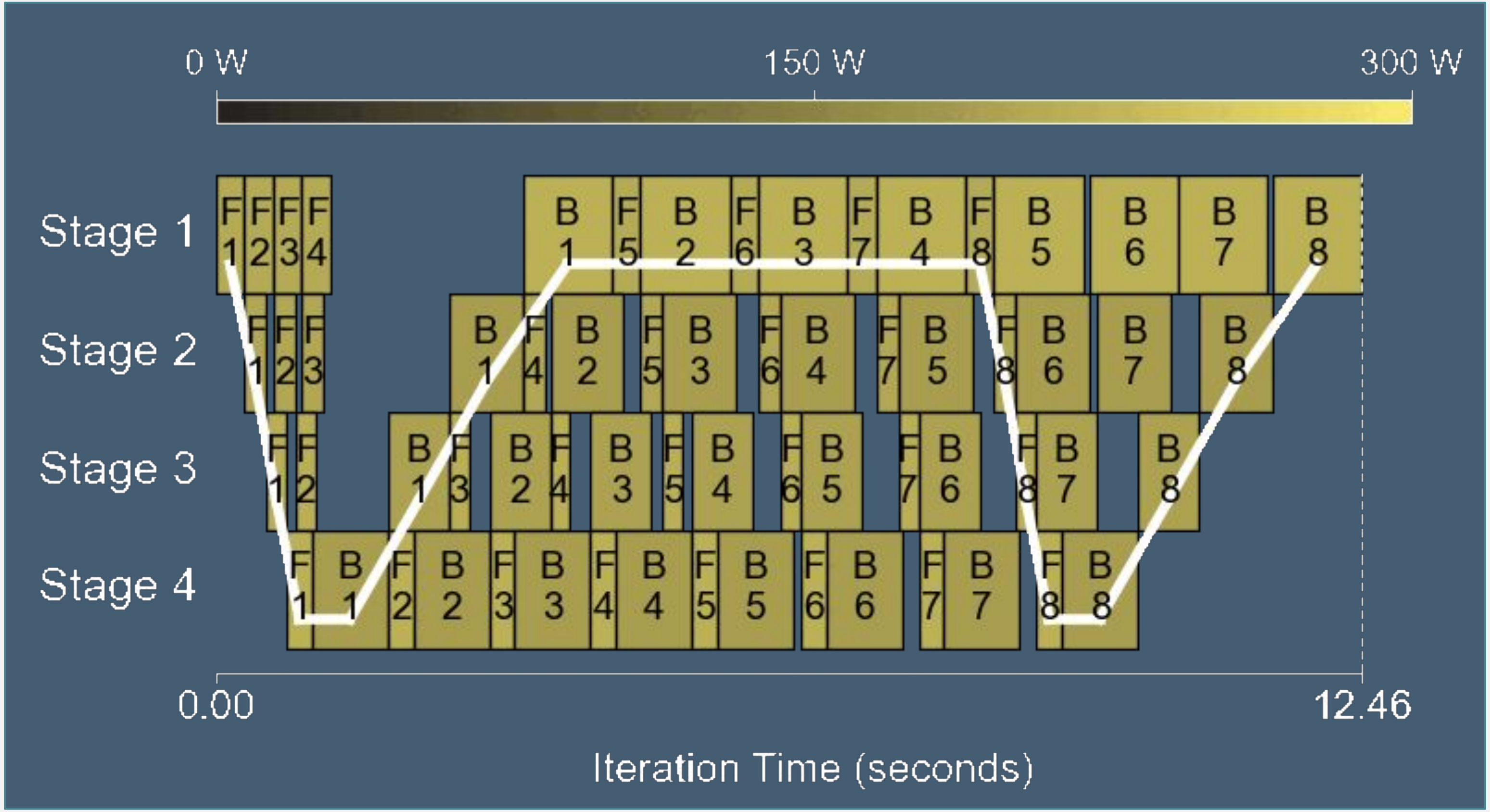


From Min Cuts to Neighbor Energy Schedule

- For **all** edges on the min cut, form a new energy schedule by reducing that computation by τ
- Repeat the whole process until the desired time is met!
- Assign frequencies such that the operation time is equal to or just faster than the new runtime



Frontier Characterization



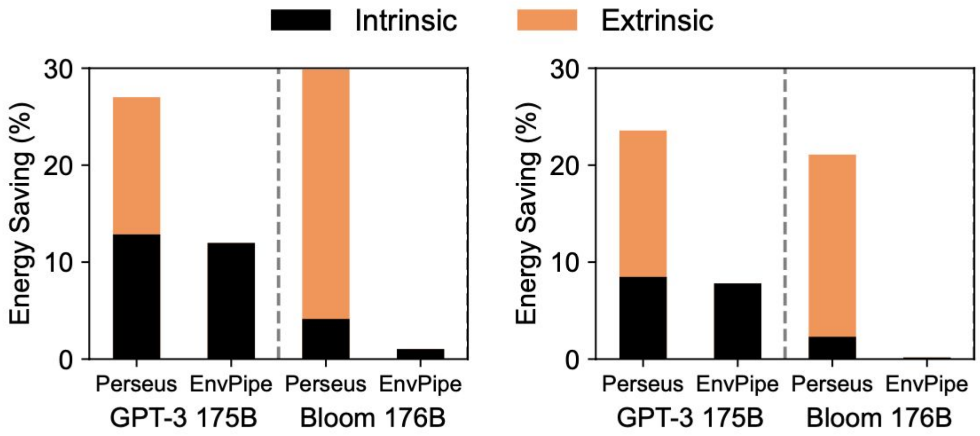
Generalizations to Other Use Cases

- **3D/Hybrid Parallelism**: Just profile 1 GPU and replicate
- **Constant-Time Operations**: Not affected by clock frequency-> treat as node with 1 freq
- **Other Pipeline Schedules**: Perseus can handle these too as long as we have a DAG representation!

Evaluation: Setup

- **Testbed**: GPU Cluster with an AMD EPYC 7513 CPU, 512 GB DRAM, 4 A40 GPUs or 2 Intel Xeon 8380 GPUs, 512 GB DRAM, 4 A100 GPUs on cloud
- **Workloads**: GPT-3, Bloom, BERT, T5, Wide-ResNet (1.3-6.7B Params on Testbed, 176 B Params Emulated)
- **Metrics**: GPU Energy Reduction (intrinsic bloat reduction without stragglers and total bloat reduction with stragglers)
- **Baseline**: EnvPipe (reduces only intrinsic bloat), Zeus (characterizes time energy tradeoff)

Evaluation: Reducing Energy Bloat



(a) A100 **(b)** A40

Figure 7. [Emulation] Energy savings breakdown with straggler slowdown 1.2 and 1,024 GPUs.

Perseus Overhead

- **Profiling**: 13 minutes on A100 workloads -> negligible

- **Algorithm Runtime**: 6.5 minutes on A100 workloads, justified due to training time dominating
  - Looking up optimal schedule given straggler time is instant

Technical Innovations

Unified Optimization Framework

- Perseus combines both **intrinsic** and **extrinsic** energy optimization in a unified approach

Graph Cut-Based Energy Scheduling

- Uses a **graph-cut algorithm** to represent each training iteration as a **Directed Acyclic Graph (DAG)**

Time-Energy Tradeoff Frontier Enabling Dynamic Response

- Perseus pre-characterizes the time-energy frontier, calculating optimal iteration times to reduce energy consumption.

Limitations of Perseus

Dependency on Straggler Awareness:

- **Reliance**: To maximize extrinsic energy savings, Perseus relies on notifications about stragglers (e.g., through power or temperature managers)
- **Implication**: Without effective straggler detection, Perseus may miss opportunities for energy savings, particularly in large-scale settings

Constraints on Hardware Support:

- **Requirement**: Perseus requires GPUs that support multiple execution speeds for dynamic frequency control
- **Impact**: Systems without frequency scaling options cannot benefit from Perseus's energy-saving mechanisms, limiting its applicability across diverse hardware setups

Thank you for listening!