ChatGPT

DRL-Based Parameter Pruning: Designing Emrakul’s Surgical Policy

The "Nervous System": State Representation

Scaling Observations: A central challenge is how Emrakul (the DRL pruning agent) can “sense” millions of weights without an infeasible observation vector. The solution is a hierarchical *Magistrate* architecture that aggregates *local* statistics into a compact global state. We wrap each small region of the network (e.g. a filter, neuron, or block of weights) in a lightweight **“Phage”** module that monitors that region’s vital metrics. Each Phage computes a local state vector summarizing key statistics – for example, it might collect the average weight magnitude, recent gradient variance, sign change frequency, Fisher information, and an “age” counter for that region. This local state is then fed through a shared small encoder network (the Phage’s policy) to compress it into an embedding. All Phages share the same encoding policy (parameter tying) so that millions of weights can be processed by a manageable number of learned parameters. The embeddings from all regions are then pooled or passed through an attention mechanism to form the **Magistrate’s** global observation. In this way, the agent sees a **compressed neural health report** instead of every raw weight, avoiding an explosion in input dimensionality.

Local Vital Signs: Instead of relying only on static metrics like magnitude, each Phage computes *dynamic* indicators of a weight’s utility (the “vital signs” of a parameter): - **Fisher Information:** An estimate of how sensitive the loss is to this weight. A low Fisher info implies the weight can change (or be pruned) with minimal effect on the output [1]. This is more context-aware than raw magnitude, since even a large weight might be unimportant if the loss gradient w.r.t. it is near zero. - **Gradient Signal-to-Noise Ratio (SNR):** Measures how “informative” a weight’s gradient is. If the gradient over minibatches is mostly noise (low SNR), the weight might be learning nothing useful [2]. A low SNR (or erratic gradient sign flips) indicates the parameter is *metabolically expensive* (causing update noise) but *functionally useless* for reducing loss. In fact, using gradient SNR as a pruning criterion dynamically identifies weights that no longer carry meaningful signal [3] [2]. - **Gradient Coherence/ Orthogonality:** The Phage can track whether a weight’s updates align with the broader optimization direction. For example, if the weight’s gradient is frequently orthogonal to the layer’s average gradient or flips sign back-and-forth, it suggests this weight is out-of-sync with learning dynamics. Such a weight might be redundantly canceling out others or oscillating without contributing to convergence. - **Activation/Utilization Patterns:** Metrics like how often a neuron saturates or how variance in its output changes can signal redundancy. A weight that rarely activates (or whose output is consistently corrected by batch-norm/others) may be pruned with little effect. - **Age and Update History:** “Age” can denote how long a weight has been effectively dormant. If a weight’s magnitude and gradient have both remained near-zero for many iterations (stagnant weights), it’s a strong candidate for removal. Conversely, a recently pruned (or grown) weight might be temporarily exempt from cutting to allow it to prove its usefulness.

Each Phage encodes these local signals into a fixed-length state vector (e.g. `[|w|, Fisher, grad_SNR, sign_flip_rate, age, ...]`). A small network ϕ_θ then produces an embedding $e_i = \phi_\theta(s_i)$ for that region. The **Magistrate** (global agent) integrates ${e_i}$ from all regions – for instance by concatenation or via a graph neural network that treats each region as a node – and decides on pruning actions. This design massively reduces the observation space by focusing on *summarized health indicators* rather than raw weights. It also enables parameter-sharing: the same learned “intuition” (the Phage policy) is applied across all parts of the network. In essence, Emrakul develops a *surgical intuition* by looking at a patient monitor of vital signs rather than micromanaging every single synapse.

The "Scalpel": Action Space & Granularity

Hierarchical Action Structure: Emrakul’s action space must be structured hierarchically to manage pruning at scale. Instead of outputting a 100-million-dimensional mask directly, the agent manipulates *macro-parameters* that indirectly determine which weights to prune: - *Continuous Threshold Control:* One approach is for the agent to output a **sedation threshold** k (or fraction) at each decision point – for example, “prune the lowest k% of weights (by importance score) in the network (or in a target layer).” This turns a single continuous action into a coordinated pruned set. The agent can adjust k over time (analogous to a sparsity schedule) to increase or decrease the aggressiveness of pruning. - *Focused Pruning Decisions:* The action can be multi-dimensional and hierarchical: e.g. first choose a particular layer or block to target, then choose how much to prune there. In practice, the agent might have a two-tier policy: a high-level action picks *where* to operate (which layer/module), and a lower-level action picks *what fraction* or which pattern of weights to remove. This sequential decision (layer -> amount) is how some RL-based pruning methods handle large models [4] [5]. For example, prior work used RL to assign layer-wise sparsity ratios by treating each layer’s compression rate as an action [6]. - *Gating and Gumbel-Softmax:* To allow fine-grained control, we can introduce continuous gating variables that the agent tweaks. For instance, imagine each weight w_{ij} has an associated gate $m_{ij}\in[0,1]$ that multiplies it ($y_{ij} = m_{ij} \cdot w_{ij}$). A *hard* pruning corresponds to $m_{ij}=0$. Instead of binary decisions, Emrakul’s action could adjust a global **temperature τ** in a Gumbel-Softmax sampling of these gates – effectively setting how “hard” or “soft” the masking is. A high τ keeps gates fuzzy (many weights only partially suppressed), whereas a low τ pushes gates to 0/1 extremes, implementing a near-binary mask. By outputting τ or similar continuous knobs, the agent indirectly controls a whole masking distribution. This trick can turn a massive binary action space into a manageable continuous one, with the stochastic gating providing coverage of many pruning combinations (akin to how differentiable mask methods use temperature to gradually enforce sparsity).

Hardware-Friendly Sparsity Constraints: In designing Emrakul’s actions, we must ensure the learned pruning pattern is aligned with real hardware efficiency. Random unstructured zeros may reduce parameter count but won’t speed up computation on standard devices [7]. We therefore bias the action space (or reward) toward *structured* sparsity: - **N:M Sparsity:** The agent’s actions can be constrained such that for every block of M weights, exactly N remain non-zero (e.g. 2:4 sparsity means in every 4 weights, 2 are kept) [8]. This introduces a regular pattern that GPUs can exploit [7]. Concretely, instead of allowing an arbitrary subset of weights to be pruned, Emrakul’s decision could be framed as *which N out of each M to keep*. That reduces the action to selecting positions within fixed blocks – a much smaller, repeating decision. The Phage local state can also be computed per block to help the agent pick which entries in the block to zero out. By repeating this pattern across the network, we guarantee dense compute kernels (like NVIDIA’s 2:4 sparse tensor cores) can accelerate the result [9] [10]. - **Group/Filter Pruning:** Alternatively, Emrakul might act at the level of entire channels or heads. The action could be to drop a set of neurons (e.g. “remove 3 filters in layer 4”). This *structured pruning* (removing whole filters) directly yields speedups on standard hardware [11]. A hierarchical policy can first decide how many channels to prune in a layer, and then which ones – akin to a two-stage action. In practice, a DRL agent can learn an optimal per-layer sparsity budget (how many filters to prune in each layer) [12] [5], and then within that, identify specific channels with low “vital signs” to remove. By grouping weights in channel/block units in both state and action, the agent naturally produces *block sparsity*, which is friendly to CPU/GPU memory access patterns [13] [14]. - **Penalty for Fragmentation:** If the agent were to propose a truly irregular mask, we can incorporate a penalty in the reward (or as a constraint) for deviation from supported patterns. For example, if not using a structured grouping in the action space, the reward could subtract points for each violation of N:M (each block that doesn't conform) or for each time a weight is pruned in isolation rather than as part of a contiguous block. This nudges the policy toward clumping pruned weights into hardware-efficient configurations. In essence, the agent must learn to "cut with a straight scalpel" – e.g. prune an entire vector lane – rather than poke random holes.

By structuring the action space hierarchically and including hardware-aware patterns, Emrakul's pruning decisions become both **scalable** and **realistic**. The agent might, for instance, output something like: *Layer 3: apply 2:4 sparsity with threshold X*, which results in a deterministic mask (pruning the smallest-magnitude half of weights in each 4-weight block for that layer). This drastically simplifies the policy output while still giving the agent flexibility to tune X or choose layers adaptively.

The "Surgery": Safe Protocols & Reversibility

Performing surgery on a "beating heart" – pruning during training – requires caution. Emrakul must learn procedures to minimize trauma and even reverse course if a cut proves harmful:

Sedation vs. Lysis (Reversible Pruning): Inspired by medical surgery, we give the agent a two-stage operation for each weight: - **Sedation (Soft Masking):** Instead of immediately severing a connection, Emrakul can *sedate* it by setting its mask m_{ij} to a small value (or zero) in the forward pass while **still retaining the weight's value and updating it in the background**. In other words, the weight's influence on the network is temporarily nullified, but it isn't truly removed – it's merely inactivated. During this sedation period, we continue to compute gradients for the weight (as if m_{ij} were 1 in backprop) so that if the agent "wakes" it up, it can reintegrate smoothly. This technique was effectively used in Dynamic Network Surgery, where a binary mask was applied to weights for inference, but gradients flowed through the unmasked weights in training [15] [16]. By doing so, pruned connections weren't permanently lost – they could be *spontaneously revived* if needed. - **Testing the Cut:** While a weight is sedated, the agent monitors the impact on the loss and other vital signs. If the network continues training stably (no significant loss spike or if any spike quickly recovers), it's evidence that the weight was indeed redundant. Emrakul can then safely proceed to **lysis**. - **Lysis (Permanent Removal):** Lysis commits the removal – the weight is physically pruned from the network (or set permanently to zero with no further updates). Memory is reclaimed and the network's structure is updated (e.g. the next layer's input dimension is decremented if an entire neuron is lysed). Lysis is irreversible within the episode, so Emrakul should only do this once confident (often after a successful sedation test).

This reversible protocol allows **safe exploration** of sparsity: Emrakul can try zeroing a parameter with minimal risk. If a sedated weight turns out to cause a sustained loss increase, the agent can simply restore its mask to 1 (wake it up) instead of having caused permanent damage. Essentially, sedation gives the network a chance to *reconfigure and compensate* for a potential removal, while keeping a life-line to undo the change. This is analogous to a surgeon clamping an artery to see the effect before permanently severing it.

Trauma-Aware Reward Signal: Designing Emrakul's reward is a delicate balance between encouraging sparsity and avoiding catastrophic drops in accuracy (the "trauma"). We need a reward R that captures both the *efficiency gains* of pruning and the *pain* inflicted: - **Efficiency Term:** A straightforward component is the negative of model size or FLOPs. For example, give a positive reward for each percentage of weights pruned or for meeting a target sparsity level. This pushes the agent toward aggressive compression. - **Performance Term:** We also include a term for validation accuracy or loss. For instance, $R_{\text{perf}} = -(L_{\text{val}})$ (negative loss) or a sparse penalty that is zero as long as accuracy remains within δ of the original. The agent thus knows that if it prunes too much and accuracy collapses, it will incur a large negative reward. - **Smoothness/Trauma Term:** To make the agent sensitive to *how* pruning affects training dynamics, we introduce a penalty for **loss spikes**. One implementation: whenever Emrakul takes an action, measure the immediate change in training loss $ \Delta L$. A sharp increase beyond a threshold incurs an immediate negative reward (signifying "ouch, that hurt"). However, if the loss recovers after a few gradient steps (indicating the network adapted), we could later add back a small positive reward – signaling "good pain" that led to eventual healing. Essentially, we treat a transient increase that quickly settles as acceptable (or even necessary for long-term gain), whereas a sustained loss increase is "bad pain". This could be done by measuring area under the loss curve after the action or the recovery time. - **Delayed Reward and Adaptation:** Because the ultimate goal is final accuracy *and* sparsity, the reward can be mostly given at the end of training (or end of an episode) to reflect the final outcome. For example, a combined metric like $R_{\text{final}} = \text{Accuracy} - \lambda \cdot \text{Sparsity}$ (higher is better) can be the main reward. But to help learning, shaped rewards at intermediate points are given as described (penalize big instant losses, reward smaller network sizes, etc.). This encourages the agent to prune in a way that the final accuracy after retraining is maximized for a given sparsity. In practice, RL-based pruning methods often define the reward as a weighted sum of accuracy and compression ratio [17], letting the agent implicitly learn the trade-off.

Formulating the reward this way teaches Emrakul an important concept: **not all pain is equal**. A brief accuracy drop that leads to a higher long-term accuracy (after fine-tuning) is actually beneficial – analogous to a patient experiencing short-term pain from surgery that improves their health. In contrast, an unrecoverable drop (network collapse) is to be avoided. The agent's policy, through trial and error, will discover strategies that maximize reward: typically these are ones that prune boldly but in a controlled, stepwise manner that the network can recover from (small, "safe" cuts first, then larger ones once confidence grows). Additionally, by incorporating a *recovery phase* in the training regimen (e.g. after each pruning action, allow a few optimization steps or even a micro fine-tuning), we can explicitly measure how well the network bounced back. The agent essentially gets feedback like "I removed those 1000 weights and the loss went up by 5%, but after 100 minibatches it dropped back to baseline – net reward: mildly negative for the spike, plus positive for smaller model." Over time, Emrakul will learn to perform **minimal-trauma surgeries** – for example, pruning weights that it predicts the network can live without (perhaps those with high gradient redundancy or support from similar weights) so that any loss increase is minor and quickly recovered.

In conjunction with sedation, the agent has a safety net: if an attempted cut causes a bad reaction, it can reverse the sedation (incur some penalty but prevent total failure) and try a different approach. This is essentially **safe exploration** in the action space of network modifications.

Emergent Strategies (The "Magic")

If successful, a DRL-driven pruning agent like Emrakul could uncover pruning strategies far more nuanced than any static heuristic. Some speculative advanced behaviors include:

- **Cycle-Based Pruning and Regrowth:** Emrakul might learn to perform iterative *prune-regrow cycles* as a way to escape local minima. For example, the agent could intentionally prune a subset of weights (introducing a temporary capacity loss), forcing the network to reroute and re-specialize remaining weights. Once the network has shifted into a new configuration, Emrakul (or its counterpart Tamiyo) could reintroduce connections (either restoring sedated weights or growing new ones) in more optimal locations. Such cycles echo the dense-sparse-dense (DSD) strategy where pruning then restoring weights led to a better minimum than the original network had [18]. By pruning and then regrowing, the agent is effectively performing an

annealing: it knocks the model out of a sharp basin and allows it to settle into a flatter (potentially better) minimum when capacity is returned [18]. A static heuristic wouldn't normally consider *re-adding* weights after pruning, but a DRL agent could discover that two steps forward, one step back yields a better long-term outcome.

- **Simulated Annealing via Surgical Perturbations:** Beyond formal cycles, Emrakul could learn to occasionally sacrifice accuracy *on purpose* to achieve a net gain later. For instance, the agent might execute a particularly aggressive pruning in an overfit layer, knowing that the ensuing drop in accuracy will spur the network to relearn with a sharper focus (and perhaps with regularization benefits). After a period of adaptation, the network might not only recover but generalize better. This resembles simulated annealing or controlled burns – short-term damage that yields long-term health. An example might be pruning an entire redundant feature detector, causing a dip in performance, but allowing new more useful features to emerge upon retraining. Heuristics typically avoid any move that immediately hurts loss, but an RL agent optimizing final reward could discover that **sometimes you must hurt to heal**. Indeed, strategies like deliberately over-pruning and then fine-tuning can occasionally surpass the original accuracy because they remove spurious connections and allow more robust ones to grow (a phenomenon observed in some extreme pruning experiments).

- **Adaptive Budgeting and Coordination:** In a multi-agent setup (with Tamiyo for growth and Emrakul for pruning under a supervisor), Emrakul might exhibit nuanced behavior like *saving budget* for when the network is most stable. For example, it could refrain from heavy pruning during periods of rapid learning (when the "patient" is unstable) and strike when a plateau is reached. It might also learn to coordinate with Tamiyo by pruning in one area to *make room* for growth in another. For instance, if a certain layer is under-utilized, Emrakul prunes it down, freeing compute budget which the supervisor can reallocate to another layer where Tamiyo adds neurons. Over time, this could yield an emergent division of labor: Emrakul carves back unused capacity and funnels the "energy" to places where learning progress is maximal. While this goes somewhat beyond single-agent RL, it highlights the potential for *policy meta-learning* – Emrakul could be trained on many "host" networks and learn a generalized sense of *where* and *when* pruning is advantageous, effectively developing a pruning policy that transfers across architectures.

In summary, a DRL-driven pruning agent can leverage **contextual, time-varying intuition** to navigate the stability-efficiency trade-off better than any static math-driven method. It watches the network's vital signs and learns the art of **surgical sparsification**: knowing not just *which* weights to cut, but *when* and *how* to cut them with minimal trauma and maximal long-term gain. Through careful state representation, a structured action space, and safe-pruning protocols, Emrakul could truly become a smart "immune system" – trimming the fat of a neural network in a way that continuously improves its health. The hope is that such an agent will discover the magic combination of moves (prune, regrow, test, and so on) that yields leaner networks *without* sacrificing – and indeed sometimes enhancing – their performance. [18] [17]

Sources:

- Le Cun et al., "Optimal Brain Damage" – initial proposal of using second-derivative (Hessian) information for pruning.
- Hassibi et al., "Optimal Brain Surgeon" – improved Hessian-based weight pruning.
- Guo et al., *Dynamic Network Surgery*, NeurIPS 2016 – introduces pruning with the ability to **recover** connections during training [15] [16].

- Siems et al., *Dynamic Pruning via Gradient Signal-to-Noise Ratio*, NeurIPS 2021 – uses gradient SNR as a dynamic criterion for pruning during training [3] [2].
- He et al., *AMC: AutoML for Model Compression*, ECCV 2018 – uses RL (via Deep Deterministic Policy Gradient) to decide layer-wise channel pruning ratios.
- Liu et al., *RL-Pruner: Reinforcement Learning for Structured Pruning*, arXiv 2023 – learns an optimal sparsity distribution across layers with a reward combining accuracy and compression [12] [17].
- NVIDIA A100 Sparse Patterns – Documentation on **2:4 N:M sparsity** showing hardware benefits of structured pruning [8] [7].
- Han et al., *DSD: Dense-Sparse-Dense Training*, ICLR 2017 – demonstrates that cycling between dense and sparse can improve final accuracy [18].

---

[1] The silence of the weights: an investigation of structural pruning strategies for Attention-based audio signal architectures
https://arxiv.org/html/2509.26207v1

[2] [3] assets.amazon.science
https://assets.amazon.science/13/93/313fb3ff48a1af7e9a2fab004811/dynamic-pruning-of-a-neural-network-via-gradient-signal-to-noise-ratio.pdf

[4] [5] [6] [12] [17] RL-Pruner: Structured Pruning Using Reinforcement Learning for CNN Compression and Acceleration
https://arxiv.org/html/2411.06463v1

[7] [8] [9] [10] [11] [13] [14] Hardware-Aware Sparse Pruning. some well-known hardware-supported… | by sandeep chowdhury | Medium
https://medium.com/@sandeepiit.ism/accelerating-ai-with-pattern-based-pruning-understanding-n-m-sparsity-d696dddbdd66

[15] [16] Reviews: Dynamic Network Surgery for Efficient DNNs
https://proceedings.neurips.cc/paper/2016/file/2823f4797102ce1a1aec05359cc16dd9-Reviews.html

[18] openreview.net
https://openreview.net/pdf?id=HyoST_9xl