One-for-All Pruning: A Universal Model for Customized Compression of Large Language Models

Rongguang Ye, Ming Tang[*]
Department of Computer Science and Engineering
and the Research Institute of Trustworthy Autonomous Systems
Southern University of Science and Technology, Shenzhen, China
yerg2023@mail.sustech.edu.cn, tangm3@sustech.edu.cn

Abstract

Existing pruning methods for large language models (LLMs) focus on achieving high compression rates while maintaining model performance. Although these methods have demonstrated satisfactory performance in handling a single user's compression request, their processing time increases linearly with the number of requests, making them inefficient for real-world scenarios with multiple simultaneous requests. To address this limitation, we propose a **Uni**veral Model for **Cu**stomized **Comp**ression (UniCuCo) for LLMs, which introduces a StratNet that learns to map arbitrary requests to their optimal pruning strategy. The challenge in training StratNet lies in the high computational cost of evaluating pruning strategies and the non-differentiable nature of the pruning process, which hinders gradient backpropagation for StratNet updates. To overcome these challenges, we leverage a Gaussian process to approximate the evaluation process. Since the gradient of the Gaussian process is computable, we can use it to approximate the gradient of the non-differentiable pruning process, thereby enabling StratNet updates. Experimental results show that UniCuCo is 28 times faster than baselines in processing 64 requests, while maintaining comparable accuracy to baselines.

1 Introduction

Large language model (LLM) compression (Ma et al., 2023; Zhu et al., 2024; Yu et al., 2024) aims to reduce the size and computational demands of pre-trained models while preserving their performance. Among commonly used techniques, pruning (Frantar and Alistarh, 2023; Yin et al., 2023) reduces the size and complexity of pre-trained models by removing less critical weights or layers while retaining their core functionality. By employing these techniques, LLMs can be deployed efficiently in resource-constrained environments, such as edge devices (Tseng et al., 2024).

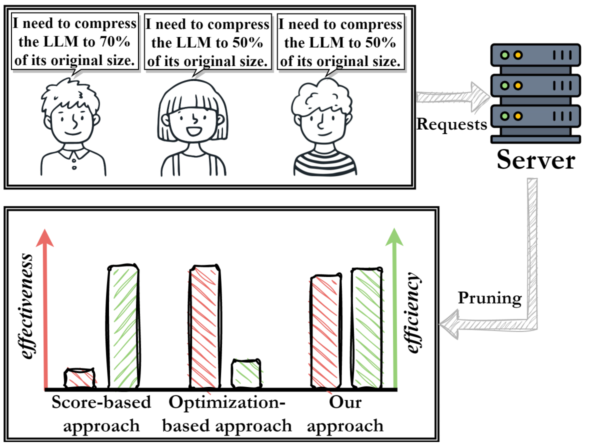

I need to compress the LLM to 70% of its original size.
I need to compress the LLM to 50% of its original size.
I need to compress the LLM to 50% of its original size.
Requests
Server
Pruning
effectiveness
efficiency
Score-based approach
Optimization-based approach
Our approach

Figure 1: A comparison of various approaches in terms of effectiveness and efficiency when providing pruning strategies for compression requests.

In practical applications, users have diverse compression requests (defined by their goals of model size reduction while preserving performance) due to the varying capabilities of their devices. Numerous LLM pruning methods have been developed to address specific compression requests (Yin et al., 2023; Kim et al., 2024). These methods can be categorized into optimization-based approaches (Sieberling et al., 2024) and score-based approaches (Men et al., 2024; Frantar and Alistarh, 2023). Optimization-based approaches frame LLM pruning as an optimization problem, utilizing heuristic algorithms (Yu and Gen, 2010) (e.g., evolutionary algorithms) to preserve the performance of the pruned LLM under a specific compression request. In Fig. 1, optimization-based approaches are highly effective in maintaining performance through iterative refinement of pruning strategies during the search process. However, their runtime efficiency in handling multiple compression requests is significantly limited, as each request necessitates an independent heuristic search, leading

[*]Corresponding Author.

to substantial time overhead. In contrast, score-based approaches calculate importance or sensitivity scores for each LLM layer, which are then used to determine the layers to prune to meet compression requests. By reusing the computed scores, the efficiency of handling multiple compression requests is enhanced. However, score-based approaches exhibit limited effectiveness in preserving performance, since the pruning strategies generated from importance scores fail to satisfy the monotonicity property (Sieberling et al., 2024). Motivated by these findings, we pose the following research question: **How can we *effectively* and *efficiently* handle multiple compression requests?**

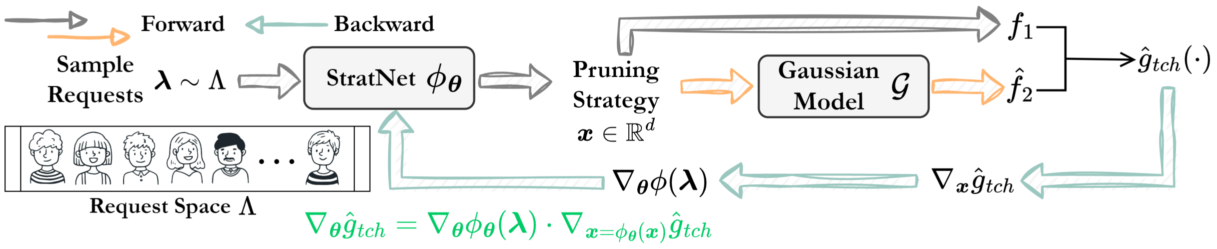
To address this problem, we propose Request-Conditional Pruning (UniCuCo) for handling multiple compression requests simultaneously. Specifically, we introduce a StratNet that maps an arbitrary compression request to its corresponding optimal pruning strategy, enabling the handling of diverse requests. It is applicable for StratNet to a wide range of pruning approaches, such as depth pruning and non-uniform pruning. The challenges in training the StratNet are twofold: First, it is difficult to balance the reduction in model size with the preservation of performance when optimizing StratNet to optimally match the user's request. Second, evaluating pruning strategies is computationally expensive. Third, applying pruning strategies (such as binary masks) to the LLM is a non-differentiable operation, which disrupts the backpropagation process and prevents StratNet from being updated using gradient-based methods. To address these challenges, we introduce the weighted Tchebycheff function in the optimization of StratNet, enabling it to effectively derive a pruning strategy that optimally aligns with the given request. Furthermore, we introduce a Gaussian process estimator to evaluate pruning strategies, significantly reducing evaluation time. Since the gradient of the Gaussian process is computable, we leverage it to restore the parts of StratNet that are disrupted in the backpropagation process. Notably, we propose an alternating update scheme where the Gaussian process and StratNet are updated in an interleaved manner. The main contributions of this paper are as follows:

- We introduce the problem of multiple request pruning for LLMs, which requires algorithms to efficiently and effectively generate pruning strategies tailored to diverse requests.
- We propose UniCuCo, a framework that maps arbitrary compression requests to tailored pruning strategies. UniCuCo includes a Gaussian process estimator, which significantly reduces evaluation time of pruning strategies and subtly solves the non-differentiable issue in UniCuCo. Additionally, we introduce an alternating scheme for updating the Gaussian process and StratNet.
- Experimental results show that UniCuCo processes 64 compression requests on the Mistral-7B model with a speed at least 28 times faster than optimization-based approaches, while maintaining comparable accuracy. Meanwhile, UniCuCo achieves an average accuracy improvement of 3% across five benchmark datasets in a non-uniform pruning scenario with 70% sparsity when compared with score-based approaches on the Mistral-7B model.

2 Related Works

2.1 Depth Pruning

Depth pruning treats each transformer block as a unit and removes entire blocks for pruning. The most common approaches are score-based methods, which compute *block importance* scores and remove those with lower scores based on a compression request. For example, Weight Subcloning (Samragh et al., 2023) is a simple yet effective technique that transfers pre-trained model knowledge to smaller variants by evaluating block importance using the ratio of ℓ_2 norms between output embeddings with and without residual connections. Shortened LLaMA (Kim et al., 2024) measures block contribution by removing each block from a pre-trained model and assessing its impact on perplexity. ShortGPT (Men et al., 2024) determines importance through cosine similarity between block inputs and outputs, where lower similarity indicates higher importance. Gromov et al. (Gromov et al., 2024) groups consecutive blocks and evaluates their importance using cosine similarity. However, according to EvoPress (Sieberling et al., 2024), score-based approaches in depth pruning are not monotonic. That is, a pruned LLM with a higher cumulative importance score does not necessarily lead to higher effectiveness in preserving performance. To address this limitation, several optimization-based approaches have been proposed for depth pruning. For example, Sheared-LLaMA (Xia et al.) introduces a mask learning phase to identify prunable components across blocks.



Forward
Backward
Sample Requests
$\boldsymbol{\lambda} \sim \Lambda$
StratNet $\phi_{\boldsymbol{\theta}}$
Pruning Strategy $\boldsymbol{x} \in \mathbb{R}^d$
Gaussian Model $\mathcal{G}$
f_1
$\hat{f}_2$
$\hat{g}_{tch}(\cdot)$
Request Space Λ
$\nabla_{\boldsymbol{\theta}}\phi(\boldsymbol{\lambda})$
$\nabla_{\boldsymbol{x}}\hat{g}_{tch}$
$\nabla_{\boldsymbol{\theta}}\hat{g}_{tch} = \nabla_{\boldsymbol{\theta}}\phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}) \cdot \nabla_{\boldsymbol{x}=\phi_{\boldsymbol{\theta}}(\boldsymbol{x})}\hat{g}_{tch}$

Figure 2: Flowchart of UniCuCo.

2.2 Non-Uniform Pruning

Non-uniform pruning is a more fine-grained pruning scenario, where each transformer block is assigned a sparsity value between 0 and 1, rather than being simply set to 0 or 1. Score-based approaches in non-uniform pruning include Wanda (Sun et al.) and SparseGPT (Frantar and Alistarh, 2023). Wandb evaluates *weight importance* by assessing their impact on the calibration dataset. Specifically, it computes the dot product between the absolute value of the parameter matrix and the ℓ_2 norm of the input to calculate the weight importance score. SparseGPT computes the Hessian matrix for the weights within each block and generates the corresponding mask matrix based on the compression request. For optimization-based approaches, He et al. (He et al., 2018) and Ashok et al. (Ashok et al., 2018) employed reinforcement learning to guide the LLM compression process. However, these approaches are hindered by high computational complexity, leading to significant time overhead when processing a single compression request. To mitigate this issue, the recent OWL method (Yin et al., 2023) improves compression efficiency by pruning LLMs using layer-wise sparsity ratios proportional to their activation outlier ratios. EvoPress (Sieberling et al., 2024) formulates compression requests as constraints and employs heuristic search to determine the sparsity of each transformer block. Once the sparsity for each block is determined by EvoPress, SparseGPT (Frantar and Alistarh, 2023) is applied to perform the sparsification of the LLM. Despite these advancements, such methods still incur significant time costs, which are proportional to the number of compression requests. Our work aims to efficiently handle multiple requests while preserving the effectiveness of the pruned LLM.

3 Methodology

In this section, we introduce UniCuCo for customized compression of LLMs. Fig. 2 illustrates the flowchart of UniCuCo. First, the core idea is to introduce a StratNet that learns to map arbitrary requests to corresponding pruning strategies (Section 3.1). However, updating the StratNet requires evaluating a large number of pruning strategies on a calibration dataset, which is a time-consuming process. Thus, we propose the use of a Gaussian process to estimate the evaluation process, thereby reducing the computational overhead of evaluating pruning strategies (Section 3.2). Finally, we introduce methods for updating StratNet and the Gaussian process (Section 3.3).

3.1 UniCuCo Framework

We consider a cloud server that provides pruning strategies $\boldsymbol{x} \in \mathbb{R}^d$ for diverse compression requests, where d is the number of LLM blocks. Each element $x_i \in [0,1]$ represents the sparsity ratio of the i-th block. If x_i is binary, it corresponds to depth pruning, whereas if x_i takes a value in the continuous range $[0,1]$, it corresponds to non-uniform pruning. Subsequently, we define the compression request and incorporate it into the pruning optimization task. Then, we introduce StratNet and discuss its optimization method.

3.1.1 Request Formulation

Each compression request is associated with **model size reduction** and **performance preservation**. To learn the pruning strategy for any given request (i.e., the Pareto front corresponding to all requests), we propose representing compression requests using $\boldsymbol{\lambda} \in \mathbb{R}^2_+$, where $\lambda_1 + \lambda_2 = 1$. Each λ_i is used to balance the trade-off between model size reduction and performance preservation. The set of all requests over these objectives defines the request space $\Lambda = \{\boldsymbol{\lambda} \in \mathbb{R}^2_+ \mid \sum_{i=1}^2 \lambda_i = 1\}$.

Given pruning strategy $\boldsymbol{x}$, we quantify the model size reduction objective as:

$$\min_{\boldsymbol{x}} f_1(\boldsymbol{x}) = 1 - \frac{\sum_{i=1}^d x_i}{d}, \quad (1)$$

where $\frac{\sum_{i=1}^d x_i}{d}$ represents the average sparsity of the pruned LLM. A smaller value of $f_1(\boldsymbol{x})$ leads to a smaller model size, as it reflects a higher sparsity.

Following (Sieberling et al., 2024), we adopt the KL-divergence between the outputs of the pruned and unpruned LLM to characterize performance preservation:

$$\min_{\boldsymbol{x}} f_2(\boldsymbol{x}) = \mathcal{D}_{\mathrm{KL}}(P_{\mathcal{M}_{\boldsymbol{x}}} \,\|\, P_{\mathcal{M}}), \qquad (2)$$

where $P_{\mathcal{M}}$ denotes the output distribution of the unpruned LLM, while $P_{\mathcal{M}_{\boldsymbol{x}}}$ is the output distribution of the pruned LLM determined by the pruning strategy $\boldsymbol{x}$. f_2 quantifies the discrepancy in model outputs on the calibration dataset $\mathcal{D}$, with smaller values indicating better performance preservation.

3.1.2 StratNet

We propose a StratNet $\phi_{\boldsymbol{\theta}}$, parameterized by $\boldsymbol{\theta}$, that maps the request $\boldsymbol{\lambda}$ to the corresponding pruning strategy $\boldsymbol{x}$, as follows:

$$\boldsymbol{x} = \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}). \qquad (3)$$

We consider optimizing StratNet with respect to two objectives, f_1 and f_2. Thus, the optimization of StratNet is formulated as a bi-objective optimization problem: $\min_{\boldsymbol{\theta}}[f_1(\boldsymbol{x}), f_2(\boldsymbol{x})]$. A straightforward approach to solving this problem is to compute a weighted sum of f_1 and f_2:

$$\min_{\boldsymbol{\theta}} g_{ws}(\boldsymbol{x} \mid \boldsymbol{\lambda}) = \min_{\boldsymbol{\theta}} \sum_{i=1}^{2} \lambda_i f_i(\boldsymbol{x}). \qquad (4)$$

As shown in Fig. 3, $g_{ws}(\cdot)$ is applicable only to convex Pareto fronts, and not to concave ones. To overcome this limitation, we propose the following weighted Tchebycheff function (Miettinen, 1999):

$$\min_{\boldsymbol{\theta}} g_{tch}(\boldsymbol{x} \mid \boldsymbol{\lambda}) = \min_{\boldsymbol{\theta}} \max_{i\in[2]} \{\lambda_i(f_i(\boldsymbol{x}) - z_i^*)\}, \quad (5)$$

where z_i^* is ideal value for objective f_i (i.e., the lower bound of f_i). The function $g_{tch}(\cdot)$ can be used to identify the optimal pruned strategy in different kinds of Pareto front.

To enable the StratNet to learn pruning strategies for all possible requests, we optimize StratNet over the entire request space Λ as follows:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\lambda}\sim\Lambda}\left[g_{tch}(\boldsymbol{x} = \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}) \mid \boldsymbol{\lambda})\right]. \qquad (6)$$

For each sampled request $\boldsymbol{\lambda}$ in Eq. (6), $f_1(\boldsymbol{x} = \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}))$ in g_{tch} can be directly computed by Eq. (1). In contrast, $f_2(\boldsymbol{x} = \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}))$ cannot be directly

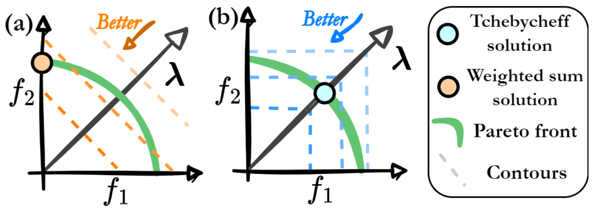


Figure 3: The optimal pruning strategy $\boldsymbol{x}$ obtained using (a) the weighted sum function and (b) the weighted Tchebycheff function under a concave Pareto front.

computed solely from $\boldsymbol{x}$. Its computation requires generating the compressed model based on $\boldsymbol{x}$ and performing inference on a calibration dataset to derive its value. This introduces two key challenges.

(I) **Significant computational overhead.** For each sampled compression request $\boldsymbol{\lambda}$, evaluating f_2 with $g_{tch}(\cdot)$ is time-consuming, as it requires the pruned LLM to perform inferences on the calibration dataset.

(II) **The feasibility of updating the StratNet.** The computation of the gradient $\nabla_{\boldsymbol{\theta}} g_{tch}$ is necessary for updating the StratNet. However, computing $\nabla_{\boldsymbol{\theta}} g_{tch}$ is challenging because $\nabla_{\boldsymbol{\theta}} f_2$ in $\nabla_{\boldsymbol{\theta}} g_{tch}$ involves the following chain rule:

$$\nabla_{\boldsymbol{\theta}} f_2(\boldsymbol{x}) = \nabla_{\boldsymbol{\theta}} \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}) \cdot \nabla_{\boldsymbol{x}} \mathcal{M}_{\boldsymbol{x}} \cdot \nabla_{\mathcal{M}_{\boldsymbol{x}}} f_2(\boldsymbol{x}), \quad (7)$$

where $\nabla_{\boldsymbol{x}} \mathcal{M}_{\boldsymbol{x}}$ cannot be computed, as the mapping from pruning strategy $\boldsymbol{x}$ to a pruned model $\mathcal{M}_{\boldsymbol{x}}$ is a non-differentiable operation. It disrupts the computational chain for Eq. (7).

3.2 Gaussian Process for Efficient Estimation

We introduce a Gaussian process denoted by $\mathcal{G}$, to compute $f_2(\boldsymbol{x})$ when solving problem (6). The key idea is to use $\mathcal{G}$ as an estimator to approximate $f_2(\boldsymbol{x})$ without requiring expensive inference on the calibration dataset for new pruning strategies. This Gaussian process can address challenges (I)–(II). We begin by presenting the Gaussian process and integrating it into our framework.

The Gaussian process $\mathcal{G}$ is defined by a mean function $\mu(\cdot)$ and a covariance function $k(\cdot,\cdot)$:

$$\hat{f}_2 \sim \mathcal{G}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x})). \qquad (8)$$

Initially, we construct a set of C observed samples $\{\boldsymbol{x}^{(c)}, f_2(\boldsymbol{x}^{(c)})\}_{c=1}^{C}$. These samples are used to train the Gaussian process, i.e., to estimate the mean function and covariance function.

Once $\mathcal{G}$ is trained, it can be used for *inference* on new pruning strategies. When a new pruning strategy $\boldsymbol{x}^{new} = \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}^{new})$ is generated by StratNet

$\phi_{\boldsymbol{\theta}}$ in problem (6), $\mathcal{G}$ computes the posterior mean $\hat{\mu}(\boldsymbol{x}^{new})$ and variance $\hat{\sigma}^2(\boldsymbol{x}^{new})$ for $\boldsymbol{x}^{new}$. The posterior mean $\hat{\mu}(\boldsymbol{x}^{new})$ serves as an estimate of $f_2(\boldsymbol{x}^{new})$, while the variance $\hat{\sigma}^2(\boldsymbol{x}^{new})$ quantifies the uncertainty of this estimate. To balance exploration (trying uncertain strategies) and exploitation (focusing on good predicted performance), we incorporate uncertainty into the estimation of f_2 for $\boldsymbol{x}^{new}$ by employing criteria such as the Lower Confidence Bound (LCB) or Upper Confidence Bound (UCB). Specifically, when applying LCB, the estimate of f_2 for $\boldsymbol{x}^{new}$ is given by

$$\hat{f}_2(\boldsymbol{x}^{new}) = \hat{\mu}(\boldsymbol{x}^{new}) + \kappa\hat{\sigma}(\boldsymbol{x}^{new}), \tag{9}$$

where $\kappa \geq 0$ is a constant used to balance posterior mean and uncertainty. Using Gaussian process to estimate f_2 significantly reduces the computation time from tens of seconds on the calibration dataset to just tens of milliseconds, achieving a thousand-fold improvement in computational efficiency and addressing challenge (I).

Meanwhile, the Gaussian process can effectively tackles challenge (II). The existence of Gaussian process ensures that $\nabla_{\boldsymbol{\theta}} f_2(\boldsymbol{x})$ can be estimated:

$$\nabla_{\boldsymbol{\theta}} \hat{f}_2(\boldsymbol{x}) \approx \nabla_{\boldsymbol{\theta}} \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}) \cdot \nabla_{\boldsymbol{x}} \mathcal{G}, \tag{10}$$

where $\nabla_{\boldsymbol{x}} \mathcal{G}$ estimates $\nabla_{\boldsymbol{x}} f_2(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \mathcal{M}_{\boldsymbol{x}} \cdot \nabla_{\mathcal{M}_{\boldsymbol{x}}} f_2(\boldsymbol{x})$. Then, when StratNet is updated, its gradient $\nabla_{\boldsymbol{\theta}} g_{tch}$ can be estimated by

$$\nabla_{\boldsymbol{\theta}} \hat{g}_{tch}(\boldsymbol{x} \mid \boldsymbol{\lambda}) = \begin{cases} \lambda_1 \nabla_{\boldsymbol{\theta}} f_1(\boldsymbol{x}) & \text{if } \lambda_1 f_1(\boldsymbol{x}) \geq \lambda_2 \hat{f}_2(\boldsymbol{x}), \\ \lambda_2 \nabla_{\boldsymbol{\theta}} \hat{f}_2(\boldsymbol{x}) & \text{if } \lambda_1 f_1(\boldsymbol{x}) < \lambda_2 \hat{f}_2(\boldsymbol{x}). \end{cases}$$

In the above equation, when $\lambda_1 f_1(\boldsymbol{x}) = \lambda_2 \hat{f}_2(\boldsymbol{x})$, a subgradient is employed, given by $\lambda_1 \nabla_{\boldsymbol{\theta}} f_1(\boldsymbol{x})$.

3.3 Updating Gaussian Process and StratNet

The accuracy of the Gaussian process in estimating f_2 is crucial for optimizing StratNet. We present StratNet's update method and a dynamic update approach for the Gaussian process. The Gaussian process is updated once per epoch while StratNet undergoes I updates per epoch. Their updates are divided into steps (A) and (B).

(A) Initializing Gaussian process and optimizing StratNet. We randomly initialize a set of N pruning strategies $X^0 = \{\boldsymbol{x}^j\}_{j=1}^N$. Next, we prune the LLM based on X^0 and compute the corresponding f_1 and f_2 values for each pruned LLM. Here, f_2 is evaluated on a calibration dataset, while f_1 measures the model size reduction. Together, these values form the set $F^0 = \{(f_1(\boldsymbol{x}^j), f_2(\boldsymbol{x}^j))\}_{j=1}^N$. Finally, we train the Gaussian process $\mathcal{G}^0$ on the pairs $\{(\boldsymbol{x}^j, f_2(\boldsymbol{x}^j))\}_{j=1}^N$ by maximizing the marginal likelihood (Rasmussen, 2003).

To optimize StratNet at the first epoch, we apply Monte Carlo sampling to estimate the expectation of requests in Eq. (11) and then use gradient descent with I steps for optimization:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \sum_{k=1}^{K} \nabla_{\boldsymbol{\theta}} \hat{g}_{tch}(\boldsymbol{x} = \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}^k) \mid \boldsymbol{\lambda}^k), \tag{11}$$

where K is the number of sampled requests and η is the learning rate.

(B) Incremental Gaussian process update and continuous StratNet update. During the t-th epoch, stage (B) selects new samples to expand the training dataset $\{\boldsymbol{X}^{t-1}, \boldsymbol{F}^{t-1}\}$ in epoch $t-1$ of the Gaussian process, thereby enhancing its prediction accuracy. Then, the Gaussian process updates on the increased training dataset, while StratNet is updated according to Eq. (11).

To increase the training dataset, we first generate a strategy candidate pool X_p^t based on StratNet. To do this, we sample a set of C requests $\{\boldsymbol{\lambda}^c\}_{c=1}^C$ from the request space Λ. The StratNet then maps these vectors $\{\boldsymbol{\lambda}^c\}_{c=1}^C$ into the candidate pool, i.e., $\boldsymbol{X}_p^t = \{\boldsymbol{x}^c = \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}^c)\}_{c=1}^C$. Next, we compute the f_1 of $\boldsymbol{X}_p^t$ and use the Gaussian process to predict the f_2 of $\boldsymbol{X}_p^t$, resulting in the objective values $\hat{\boldsymbol{F}}_p^t = (f_1(\boldsymbol{X}_p^t), \hat{f}_2(\boldsymbol{X}_p^t))$.

To select a subset from candidate set $\{\boldsymbol{X}_p^t, \hat{\boldsymbol{F}}_p^t\}$ that provides the maximum benefit to Gaussian process training, we use hypervolume (HV) (Guerreiro et al., 2021) to assess the quality of the objective set $\boldsymbol{F}$. HV is calculated by the area enclosed between each point in $\boldsymbol{F}$ and a predefined reference point $\boldsymbol{r}$:

$$\mathcal{H}_{\boldsymbol{r}}(\boldsymbol{F}) = \{\boldsymbol{a} \in \mathbb{R}^2 \mid \exists \boldsymbol{f} \in \boldsymbol{F}, \boldsymbol{f} \leq \boldsymbol{a} \leq \boldsymbol{r}\}, \tag{12}$$

where a larger value of $\mathcal{H}_r(\boldsymbol{F})$ reflects a higher quality of the set F. The key in improving the Gaussian process lies in measuring the improvement brought by adding the selected subset to the $(t-1)$-th epoch training dataset $\{\boldsymbol{X}^{t-1}, \boldsymbol{F}^{t-1}\}$. Thus, we identify a subset $\{\boldsymbol{X}_s^t, \hat{\boldsymbol{F}}_s^t\}$ from $\{\boldsymbol{X}_p^t, \hat{\boldsymbol{F}}_p^t\}$ with the largest hypervolume improvement ($\mathcal{HI}$) for $\mathcal{H}_{\boldsymbol{r}}(\boldsymbol{F}^{t-1})$, as follows:

$$\mathcal{HI}(\hat{\boldsymbol{F}}_s^t) = \mathcal{H}_{\boldsymbol{r}}(\boldsymbol{F}^{t-1} \cup \hat{\boldsymbol{F}}_s^t) - \mathcal{H}_{\boldsymbol{r}}(\boldsymbol{F}^{t-1}). \tag{13}$$

Based on Eq. (13), $\boldsymbol{X}_s^t = \arg\max_{\boldsymbol{X}_s^t} \mathcal{HI}(\hat{\boldsymbol{F}}_s^t)$. To add the selected $\boldsymbol{X}_s^t$ to the $(t-1)$-th epoch

Sparsity	Method	Mistral-7B					Llama-3-8B				
		Wiki2 (↓)	C4 (↓)	FW (↓)	**Avg. (↓)**	**Latency (↓)**	Wiki2 (↓)	C4 (↓)	FW (↓)	**Avg. (↓)**	**Latency (↓)**
0%	Dense	4.82	7.72	6.41	6.32	–	5.54	8.80	7.62	7.32	–
12.5%	Cosine (Window)	7.19	10.18	8.39	8.59	8s	13.21	19.56	14.27	15.68	9s
	EvoPress	**5.74**	**9.07**	**7.43**	**7.41**	26.7m	7.68	12.33	10.20	10.07	15.4m
	ShortGPT	7.19	10.18	8.39	8.59	**<1s**	13.21	19.56	14.27	15.68	**<1s**
	Weight Subcloning	7.19	10.18	8.39	8.59	**<1s**	13.21	19.56	14.27	15.68	**<1s**
	UniCuCo	5.93	9.39	7.67	7.66	**<1s**	**7.42**	**12.03**	**9.80**	**9.75**	**<1s**
25%	Cosine (Window)	34.94	33.7	15.08	27.91	15s	5527.47	11588.16	2388.11	6501.25	14s
	EvoPress	**10.35**	**13.44**	**10.63**	**11.47**	26.2m	**14.77**	21.30	16.74	17.60	19.6m
	ShortGPT	43.26	40.16	29.29	37.57	**<1s**	5527.47	11588.16	2388.11	6501.25	**<1s**
	Weight Subcloning	43.26	40.16	29.29	37.57	**<1s**	5527.47	11588.16	2388.11	6501.25	**<1s**
	UniCuCo	13.73	17.2	13.53	14.82	**<1s**	15.05	**20.78**	**16.18**	**17.34**	**<1s**
37.5%	Cosine (Window)	1038.98	2362	1013.9	1471.62	15s	64402.73	13833.98	3908.76	27381.82	17s
	EvoPress	**31.91**	**30.86**	**22.47**	**28.41**	25.7m	**66.21**	**80.48**	**53.82**	**66.84**	15.4m
	ShortGPT	2899.74	2327.1	1023.7	2083.50	**<1s**	64402.73	13833.98	3908.76	27381.82	**<1s**
	Weight Subcloning	2899.74	2327.1	1023.7	2083.50	**<1s**	64402.73	13833.98	3908.76	27381.82	**<1s**
	UniCuCo	42.48	36.8	25.34	34.87	**<1s**	76.89	98.54	55.86	77.10	**<1s**
50%	Cosine (Window)	3410.85	1950.6	1695.4	2352.29	10s	2054.46	1116.51	692.89	1287.95	12s
	EvoPress	4148.65	2943.6	2937.8	3343.32	26.3m	**496.86**	**396.78**	**261.37**	**385.00**	13.0m
	ShortGPT	2423.38	2135.4	1104.9	1887.89	**<1s**	1664.06	1739.99	1622.69	1675.58	**<1s**
	Weight Subcloning	2423.38	2135.4	1104.9	1887.89	**<1s**	1664.06	1739.99	1622.69	1675.58	**<1s**
	UniCuCo	**235.08**	**148.85**	**120.33**	**168.09**	**<1s**	983.97	632.06	447.28	687.77	**<1s**
62.5%	Cosine (Window)	8663.29	7568.5	8644.3	8292.01	8s	6552.93	2756.67	2839.64	**4049.75**	9s
	EvoPress	3629.51	3039.1	2597.9	3088.83	28.1m	**4711.99**	4041.00	4036.07	4263.02	15.2m
	ShortGPT	12539.6	10536	4755.3	9276.92	**<1s**	56522.08	23863.46	12350.08	30911.87	**<1s**
	Weight Subcloning	12539.6	10536	4755.3	9276.92	**<1s**	56522.08	23863.46	12350.08	30911.87	**<1s**
	UniCuCo	**1846.28**	**1170.4**	**971.79**	**1329.49**	**<1s**	8405.46	**2173.26**	**1845.45**	4141.39	**<1s**

Table 1: Depth pruning results of various methods across five sparsity levels, evaluated by perplexity (PPL) and averaged PPL. Latency refers to the time required to handle a single compression request. The best results are highlighted in bold, while the second-best results are underlined.

training dataset $\{\boldsymbol{X}^{t-1}, \boldsymbol{F}^{t-1}\}$ of the Gaussian process, we need to evaluate $\boldsymbol{X}_s^t$ on the calibration dataset, as f_2 of $\boldsymbol{X}_s^t$ is still estimated by the Gaussian process. To do this, we obtain the pruned LLMs based on $\boldsymbol{X}_s^t$, and compute their corresponding f_1 and f_2 values, forming $\boldsymbol{F}_s^t$. The training dataset at t-epoch is represented as $\{\boldsymbol{X}^t, \boldsymbol{F}^t\} = \{\boldsymbol{X}^{t-1} \cup \boldsymbol{X}_s^t, \boldsymbol{F}^{t-1} \cup \boldsymbol{F}_s^t\}$. The Gaussian process is then updated on $\{\boldsymbol{X}^t, \boldsymbol{F}^t\}$.

Afterwards, StratNet performs I steps of Eq. (11) in t-epoch. The pseudocode of UniCuCo is given in Algorithm 1 of Appendix.

4 Experiments

In this section, we validate the effectiveness and efficiency of our proposed ReCoP against state-of-the-art baselines in both depth pruning and non-uniform pruning scenarios. We further analyze the impact of different scalarization functions (i.e. Eqs. (4), (5) and others) in Appendix B.3.

4.1 Experimental Setups

Baselines. For the depth pruning scenario, where the pruning strategy for each layer is represented by binary values (0 and 1), we compare our approach with several baselines. These include the optimization-based method EvoPress (Sieberling et al., 2024), as well as score-based methods such as ShortGPT (Men et al., 2024), Weight Subcloning (Samragh et al., 2023), and Sliding Window Cosine Similarity (referred toabbreviated as Cosine (Window)) (Gromov et al., 2024). **Evaluation.** All competitive methods use Fineweb-Edu (FW) (Penedo et al., 2024) as the calibration data. We evaluate *perplexity* on the WikiText-2 (Wiki2) (Merity et al., 2016) and C4 (Raffel et al., 2020) datasets to measure the performance of pruned LLMs. Additionally, we assess *accuracy* on zero-shot tasks across a range of datasets, including WinoGrande (Sakaguchi et al., 2021), PiQA (Tata and Patel, 2003), HellaSwag (Zellers et al., 2019), and both ARC-easy and ARC-challenge (Clark et al., 2018), using the LM Eval Harness (Gao et al., 2021).

4.2 Depth Pruning Results

From the depth pruning results in Table 1, two key conclusions can be drawn: (I) Our proposed UniCuCo outperforms the three score-based methods by delivering pruning strategies in less than one second, similar to the speed of the two fastest score-based methods. Additionally, UniCuCo significantly improves average perplexity, especially as model sparsity increases. (II) UniCuCo achieves competitive results compared

Sparsity	Method	Wiki2 (↓)	C4 (↓)	ArcC (↑)	ArcE (↑)	HS (↑)	PiQA (↑)	WG (↑)	**Avg. (↑)**	**Latency (↓)**
0%	Dense	4.82	7.72	48.90	79.60	60.9	80.30	73.90	68.72	–
50%	OWL	5.69	8.94	43.90	**76.90**	55.4	78.50	70.30	65.00	40m
	EvoPress	**5.48**	**8.69**	**44.88**	76.85	**56.46**	**79.16**	**71.35**	**65.74**	122m
	Uniform	5.68	8.93	43.70	76.70	55.70	78.40	71.00	65.10	**<1s**
	UniCuCo	5.65	8.95	44.11	76.73	55.66	78.40	71.27	65.23	**<1s**
60%	OWL	7.50	11.34	38.50	71.90	46.90	75.10	70.20	60.52	40m
	EvoPress	**7.12**	**10.91**	38.05	72.56	**49.91**	**76.01**	68.98	61.10	121m
	Uniform	7.78	11.86	38.00	72.40	49.40	75.00	69.30	60.82	**<1s**
	UniCuCo	7.44	11.46	**39.33**	**72.90**	**49.91**	75.79	**69.93**	**61.57**	**<1s**
70%	OWL	17.22	21.66	27.90	62.60	38.60	67.00	63.50	51.92	40m
	EvoPress	**9.73**	**14.63**	**33.45**	**67.13**	**43.91**	**72.63**	**65.27**	**56.48**	120m
	Uniform	23.08	30.03	27.10	60.90	36.10	65.90	59.40	49.88	**<1s**
	UniCuCo	15.88	22.08	29.10	64.27	39.05	69.04	62.90	52.87	**<1s**

Table 2: Non-uniform pruning results on the Mistral-7B, evaluated at three sparsity levels, with perplexity for Wiki2 and C4 datasets, and the average zero-shot accuracy (Avg.) across the ArcC, ArcE, HS, PiQA, and WG datasets.

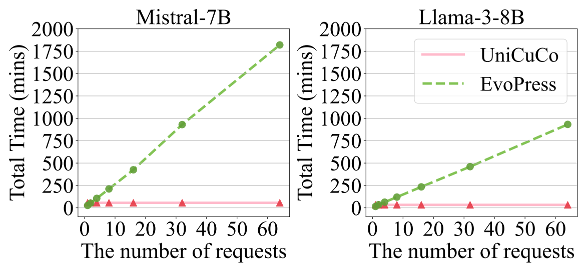


Figure 4: The comparison of total time for generating pruning strategies between UniCuCo and EvoPress as the number of requests increases.

to optimization-based EvoPress, while maintaining significantly shorter latency. Although EvoPress achieves the best perplexity in six out of ten cases across different sparsities and models, it requires approximately 13 to 26 minutes to compute the pruning strategy for each request. In contrast, our proposed UniCuCo takes less than one second in handing each request, while achieving the best results in four out of ten cases. This demonstrates that UniCuCo not only offers fast inference but also remains highly competitive in effectiveness.

We further compare the total time overhead (including both model training time and latency) of UniCuCo and EvoPress in Fig. 4. The results show that as the number of users increases, UniCuCo outperforms EvoPress in total time overhead. Specifically, on Mistral-7B, UniCuCo achieves 56 times more efficiently than EvoPress when scaling to 64 requests. This is because, once the StratNet in UniCuCo is trained, it can generate pruning strategies for any request, whereas EvoPress requires re-searching for each request.

4.3 Non-Uniform Pruning Results

In Table 2, due to the finer granularity of pruning in Non-Uniform Pruning, the distinction between algorithms is less noticeable at low sparsity, while it becomes more significant at 70% sparsity. In addition, Table 2 shows that our proposed UniCuCo not only achieves lower latency but also outperforms Uniform by 0.1%, 0.7%, and 3% in average accuracy across three sparsities. Furthermore, while EvoPress achieves the best average accuracy in two out of three sparsities, it comes at a prohibitively high time cost for handing single request. Although OWL is more time-efficient than EvoPress, its ac-

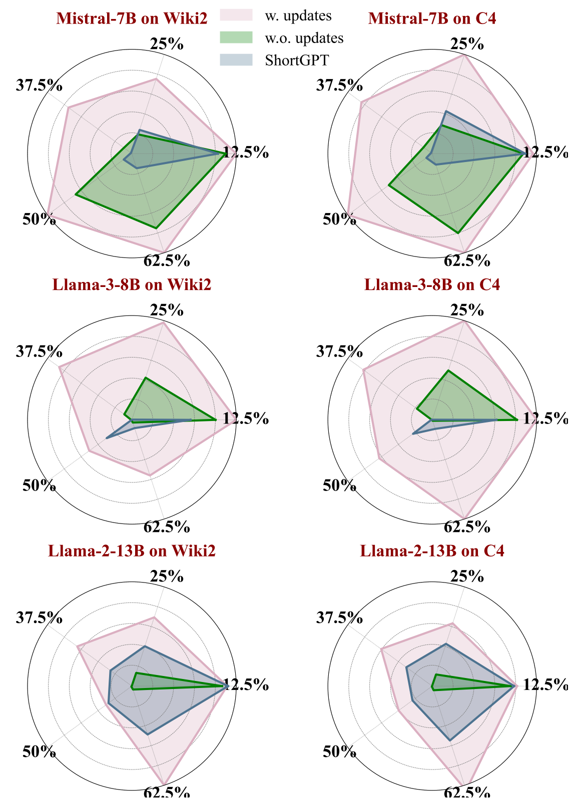


Figure 5: The impact of Gaussian process updates on depth pruning, evaluated using adjusted perplexity (higher values indicate better effectiveness).

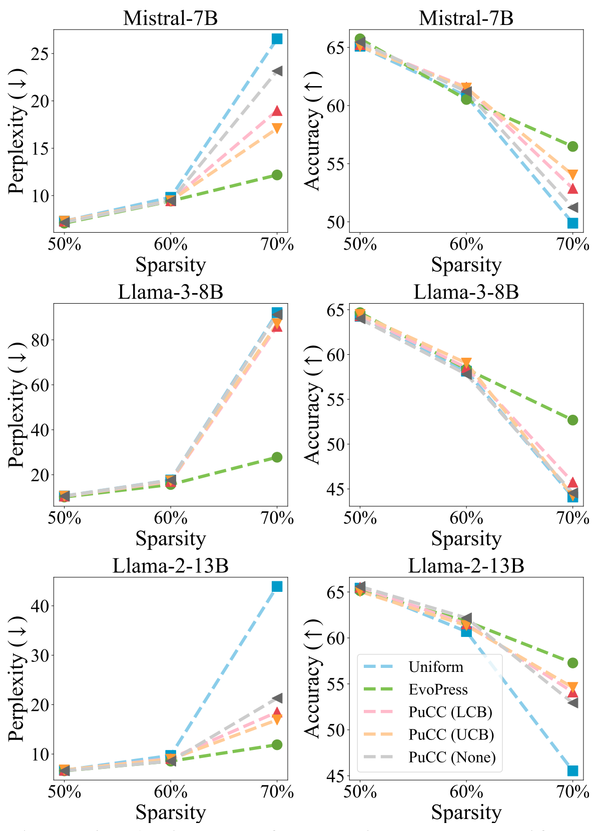


Figure 6: The impact of uncertainty on non-uniform pruning across three large models, evaluated by averaged perplexity and averaged zero-shot accuracy.

curacy is lower by 0.74%, 0.68%, and 4.56% at the three sparsity levels, respectively. In contrast, our proposed UniCuCo achieves a favorable balance, being approximately 2400 times faster than OWL and 7000 times faster than EvoPress per request in terms of efficiency, while outperforming OWL by an average of 0.6% in accuracy. We provide additional results for non-uniform pruning on Llama-3-8B and Llama-2-13B in Appendix B.2.

4.4 Effects of Gaussian Process Updates

Fig. 5 presents the pruning effectiveness based on normalized perplexity, comparing results with and without Gaussian Process updates. Results show that when the Gaussian Process is not dynamically updated, pruning effectiveness significantly declines compared to when updates are applied. Notably, on Llama-2-13, the absence of Gaussian Process updates leads to effectiveness that is lower than that of the score-based method, ShortGPT. This outcome is intuitive, as StratNet's performance relies on the Gaussian Process's estimation of f_2. Insufficient training samples hinder this estimation, thereby degrading the quality of the pruning strategies generated by the StratNet.

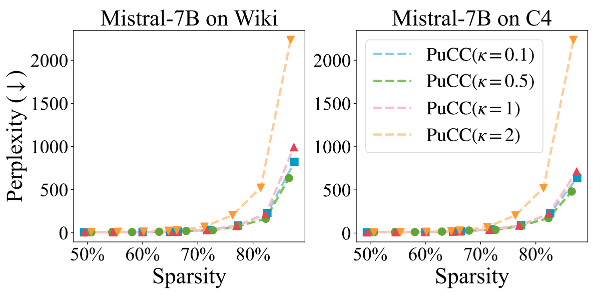


Figure 7: Effect of κ on Mistral-7B across two datasets.

4.5 Effects of Uncertainty Estimates

Recall that in Eq. (9), the prediction of pruning strategies is guided the uncertainty provided by the Gaussian process. Fig. 6 compares UniCuCo with LCB, UCB, and no uncertainty (None) against Uniform and EvoPress. An interesting observation is that ignoring uncertainty generally leads to worse effectiveness, but it still outperforms Uniform. In contrast, UniCuCo incorporating uncertainty, such as LCB and UCB, delivers superior effectiveness compared to UniCuCo (None). This is intuitively reasonable, as LCB and UCB balance exploiting well-understood regions with exploring areas of higher uncertainty, thereby improving overall effectiveness. Additionally, we show the results on the impact of the uncertainty weight κ in Fig. 7. The results indicate that variations in κ within the range of 0.1 to 1 have a marginal effect. However, when $\kappa = 2$, the pruning perplexity deteriorates as sparsity increases. This is because a larger κ causes the Gaussian process to overly emphasize uncertainty, neglecting the predictive mean.

5 Conclusion

In this paper, we proposed UniCuCo, an efficient method for handling multiple compression requests while preserving effectiveness. UniCuCo contains a StratNet that learns to map any given request to an optimal compression strategy. To overcome the challenges of high computational cost and gradient incomputability in updating the StratNet, we employ a Gaussian process to approximate updating process, thereby enabling effective learning of the StratNet. Experimental findings indicate that UniCuCo is at least 28 times more efficient than optimization-based methods for processing 64 compression requests. Additionally, it achieves 3% higher averaged accuracy in a non-uniform pruning scenario with 70% sparsity when compared with score-based methods.

Limitations

Our work currently applies UniCuCo to LLMs with 54 to 80 transformer blocks, ranging from 7B to 13B parameters. The effectiveness of UniCuCo relies on the training of the Gaussian process. As the size of LLMs increases, with hundreds of transformer blocks, the fitting space for Gaussian process training expands, and the pruning strategy dimension in the request space also increases. In this context, the effectiveness of UniCuCo requires further analysis and validation. Additionally, while our work addresses multiple compression requests for a single LLM, a more complex and realistic scenario involves handling multiple compression requests for multiple LLMs. These aspects will be explored in future research.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62202214 and Guangdong Basic and Applied Basic Research Foundation under Grant 2023A1515012819.

References

Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. 2018. N2n learning: Network to network compression via policy gradient reinforcement learning. In *International Conference on Learning Representations*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 10:8–9.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. 2024. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*.

Andreia P Guerreiro, Carlos M Fonseca, and Luís Paquete. 2021. The hypervolume indicator: Computational problems and algorithms. *ACM Computing Surveys (CSUR)*, 54(6):1–42.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*, 11.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Kaisa Miettinen. 1999. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Carl Edward Rasmussen. 2003. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Mohammad Samragh, Mehrdad Farajtabar, Sachin Mehta, Raviteja Vemulapalli, Fartash Faghri, Devang Naik, Oncel Tuzel, and Mohammad Rastegari. 2023. Weight subcloning: direct initialization of transformers using larger pretrained ones. *arXiv preprint arXiv:2312.09299*.

Oliver Sieberling, Denis Kuznedelev, Eldar Kurtic, and Dan Alistarh. 2024. Evopress: Towards optimal dynamic model compression via evolutionary search. *arXiv preprint arXiv:2410.14649*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Sandeep Tata and Jignesh M Patel. 2003. Piqa: An algebra for querying protein data sets. In *15th International Conference on Scientific and Statistical Database Management, 2003.*, pages 141–150. IEEE.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations*.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Jaiswal, Mykola Pechenizkiy, Yi Liang, et al. 2023. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. *arXiv preprint arXiv:2310.05175*.

Xinjie Yu and Mitsuo Gen. 2010. *Introduction to evolutionary algorithms*. Springer Science & Business Media.

Zhongzhi Yu, Zheng Wang, Yuhan Li, Ruijie Gao, Xiaoya Zhou, Sreenidhi Reddy Bommu, Yang Zhao, and Yingyan Lin. 2024. Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2024. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577.

A Experimental Details

Baselines. We provide detailed descriptions of the four baseline methods used for comparison in depth pruning as follows:

- **ShortGPT** (Men et al., 2024): Blocks are scored based on the average cosine similarity between their input and output embeddings, including the residual stream.
- **Weight Subcloning** (Samragh et al., 2023): Blocks are scored using the ratio $\frac{||\mathcal{M}(\mathcal{E})||}{||\mathcal{M}(\mathcal{E})+\mathcal{E}||}$, where $\mathcal{E}$ is the input embedding and $\mathcal{M}(\mathcal{E})$ is the output of block, excluding the residual stream.
- **Sliding Window Cosine Similarity** (Gromov et al., 2024): Sets of consecutive blocks are scored based on the cosine similarity between the embeddings before and after the blocks, including the residual stream.
- **EvoPress** (Sieberling et al., 2024): Determines whether to discard blocks under given compression constraints using an evolutionary algorithm.

For non-uniform pruning, all baselines and our proposed UniCuCo adopt SparseGPT (Frantar and Alistarh, 2023) as a fast and efficient one-shot layer pruning framework. SparseGPT generates sparsified blocks with varying sparsity levels across layers. The following baselines focus on searching for the optimal sparsity level for each layer:

- **Uniform**: Directly set a uniform sparsity level for all layers and extract the corresponding sparse model generated by SparseGPT.
- **OWL** (Yin et al., 2023): OWL uses Layer Outlier Distribution (LOD) metric as a measure of layer saliency, and computes a sparsity profile that is weighted by LOD.

For OWL, we used the same hyperparameter grid as the original work and took the configuration yielding the best perplexity for each model. Notably, EvoPress can also be applied to non-uniform pruning.

Hyperparameters. Following the setup in (Sieberling et al., 2024), the Calibration tokens and Evaluation tokens for the benchmark datasets are set to 524288. The maximum number of tokens that Mistral 7B and Llama-3-8B can process at once is 8192, while Llama-2-13B can process up to 4096 tokens. StratNet employs a multi-layer perceptron (MLP) architecture for training. In the process of increasing the training set in Gaussian model training based on hypervolume contribution, the candidate pool size is set to $C = 2240$. In each epoch, 10 new samples are selected and incorporated into the training set. To ensure a fair comparison, we set both EvoPress and UniCuCo to run for 50 epochs (corresponding to $T = 50$ in line 3 of the algorithm 1). For the StratNet in UniCuCo, the number of iterations in each epoch is set to 1000 (corresponding to $I = 1000$ in line 6 of the algorithm 1) with a learning rate of 1e-3. The Matérn 5/2 kernel is used in the Gaussian process.

Hardware Details. All experiments are conducted on a server running Ubuntu 22.04.5 LTS, equipped with an Intel® Xeon® Platinum 8383C CPU @ 2.70GHz and two NVIDIA L40 GPUs (46GB RAM each). All the implementations are performed using the PyTorch framework.

Algorithm 1 UniCuCo algorithm

```
1: Input: StratNet $\phi_{\boldsymbol{\theta}}$
2: // Initialize the parameters $\boldsymbol{\theta}$ and the training dataset $\{X^0, F^0\}$ for the Gaussian process $\mathcal{G}$
3: for $t = 0$ to $T$ do
4:     Training $\mathcal{G}$ on $\{X^t, F^t\}$;
5:     Sampled requests $\{\boldsymbol{\lambda}^k\}_{k=1}^K \sim \Lambda$;
6:     for $i = 0$ to $I$ do
7:         // StratNet updating
8:         Update $\phi_{\boldsymbol{\theta}}$ by Eq. (11);
9:     end for
10:    // Gaussian process updating
11:    Generating a pruning strategy pool $X_p^t = \{\boldsymbol{x}^c = \phi_{\boldsymbol{\theta}}(\boldsymbol{\lambda}^c)\}_{c=1}^C$;
12:    Select a subset $\{X_s^t, F_s^t\}$ based on Eq. (13);
13:    $\{X^t, F^t\} \leftarrow \{X^{t-1} \cup X_s^t, F^{t-1} \cup F_s^t\}$;
14: end for
15: Return: $\phi_{\boldsymbol{\theta}}$
```

B Additional Experiments

B.1 Results of UniCuCo on Depth Pruning for LlaMA-2-13B

Table 3 shows that our proposed UniCuCo responds to compression requests in less than one second and outperforms both ShortGPT and Cosine (Window) methods in terms of perplexity, which also respond within one second. Moreover, as sparsity increases, UniCuCo not only achieves competitive

Sparsity	Method	Wiki2 (↓)	C4 (↓)	FW (↓)	**Avg. (↓)**	**Latency (↓)**
0%	Dense	4.57	6.45	5.84	5.62	–
12.5%	Cosine (Window)	5.67	8.09	6.97	6.91	16s
	EvoPress	**5.42**	**7.65**	**6.62**	**6.56**	29m
	ShortGPT	5.88	8.36	7.19	7.14	**<1s**
	Weight Subcloning	5.97	8.46	7.26	7.23	**<1s**
	UniCuCo	5.94	8.07	6.94	6.98	**<1s**
25%	Cosine (Window)	8.99	12.57	10.34	10.63	24s
	EvoPress	**7.20**	**9.93**	**8.32**	**8.48**	28.6m
	ShortGPT	17.91	19.89	15.73	17.84	**<1s**
	Weight Subcloning	17.91	19.89	15.73	17.84	**<1s**
	UniCuCo	10.39	13.40	10.14	11.31	**<1s**
37.5%	Cosine (Window)	95.98	72.35	50.46	72.93	24s
	EvoPress	**13.26**	**16.69**	**13.00**	**14.32**	26.8m
	ShortGPT	52.28	46.61	35.24	44.71	**<1s**
	Weight Subcloning	52.28	46.61	35.24	44.71	**<1s**
	UniCuCo	20.49	23.66	17.74	20.63	**<1s**
50%	Cosine (Window)	1124.49	649.33	316.09	696.64	17s
	EvoPress	**52.06**	**35.75**	**28.48**	**38.76**	28.5m
	ShortGPT	187.99	165.59	132.47	162.02	**<1s**
	Weight Subcloning	187.99	165.59	132.47	162.02	**<1s**
	UniCuCo	170.34	97.43	76.79	114.85	**<1s**
62.5%	Cosine (Window)	160231.22	5219.97	4944.90	56798.70	13s
	EvoPress	4437.52	3945.83	3930.84	4104.73	23.5m
	ShortGPT	1204.75	864.48	622.41	897.23	**<1s**
	Weight Subcloning	1204.75	864.48	622.41	897.23	**<1s**
	UniCuCo	**588.07**	**451.34**	**383.87**	**474.43**	**<1s**

Table 3: Depth pruning results of Llama-2-13B across five sparsity levels, evaluated by validation perplexity (PPL). Avg. represents the averaged perplexity of three datasets.

Sparsity	Method	Wiki2 (↓)	C4 (↓)	ArcC (↑)	ArcE (↑)	HS (↑)	PiQA (↑)	WG (↑)	**Avg. (↑)**	**Latency (↓)**
0%	Dense	5.54	7.10	50.40	80.10	60.20	79.70	72.60	68.60	–
50%	OWL	8.13	13.12	43.80	75.80	54.00	75.70	**72.20**	64.30	30m
	EvoPress	**7.64**	**12.53**	**43.94**	**76.18**	**54.92**	76.17	72.14	**64.67**	139m
	Uniform	8.05	13.07	43.60	75.70	54.20	76.10	71.70	64.26	**<1s**
	UniCuCo	7.97	12.96	43.90	75.84	54.30	**76.28**	71.27	64.32	**<1s**
60%	OWL	**12.37**	**18.53**	**38.00**	**70.30**	**47.70**	72.10	68.50	**59.22**	30m
	EvoPress	**12.37**	18.92	36.01	67.34	46.45	**72.91**	**69.14**	58.37	138m
	Uniform	13.86	21.43	35.20	69.70	45.60	72.20	68.00	58.14	**<1s**
	UniCuCo	13.29	20.63	36.18	69.91	46.03	72.03	68.51	58.53	**<1s**
70%	OWL	48.07	52.32	27.00	54.90	36.60	65.10	58.60	48.44	30m
	EvoPress	**24.18**	**31.38**	**30.20**	**61.95**	**40.03**	**68.72**	**62.51**	**52.68**	138m
	Uniform	85.84	98.35	22.70	49.90	31.40	62.10	54.40	44.10	**<1s**
	UniCuCo	75.12	96.82	23.46	53.66	32.36	63.11	56.20	45.76	**<1s**

Table 4: Non-uniform pruning results of various methods on the Llama-3-8B model, evaluated at three sparsity levels with validation perplexity (PPL) and zero-shot accuracy. Avg. represents the averaged accuracy of five datasets.

results compared to the state-of-the-art method EvoPress but also demonstrates a significant advantage in terms of the time overhead required to handle compression requests over EvoPress.

Sparsity	Method	Wiki2 (↓)	C4 (↓)	ArcC (↑)	ArcE (↑)	HS (↑)	PiQA (↑)	WG (↑)	**Avg. (↑)**	**Latency (↓)**
0%	Dense	4.57	6.45	49.23	77.48	79.37	80.47	72.22	71.75	–
50%	OWL	5.61	8.02	**45.22**	**77.23**	56.20	77.26	72.85	**65.75**	114m
	EvoPress	**5.45**	**7.75**	42.75	76.85	**56.49**	77.69	71.90	65.14	139m
	Uniform	5.54	7.90	43.17	76.98	55.98	77.86	73.09	65.42	**<1s**
	UniCuCo	5.53	7.89	43.26	76.94	56.02	**77.91**	**73.40**	65.51	**<1s**
60%	OWL	7.33	10.08	**39.85**	72.69	**51.08**	75.35	69.85	61.76	98m
	EvoPress	**7.15**	**9.98**	39.25	**73.11**	50.43	**75.52**	**70.74**	**61.81**	138m
	Uniform	8.14	11.29	38.14	72.05	48.83	74.59	69.85	60.69	**<1s**
	UniCuCo	7.60	10.61	39.33	**73.11**	49.92	74.70	70.56	61.52	**<1s**
70%	OWL	14.57	17.95	31.31	65.28	41.53	70.57	67.09	55.16	121m
	EvoPress	**10.24**	**13.52**	**33.87**	**69.19**	**44.31**	**71.76**	**67.25**	**57.28**	138m
	Uniform	40.33	47.5	24.06	52.19	32.17	62.13	57.14	45.54	**<1s**
	UniCuCo	15.85	21.13	31.23	65.66	39.71	69.31	64.64	54.11	**<1s**

Table 5: non-uniform pruning results of various methods on the Llama-2-13B model, evaluated at three sparsity levels with validation perplexity (PPL) and zero-shot accuracy.

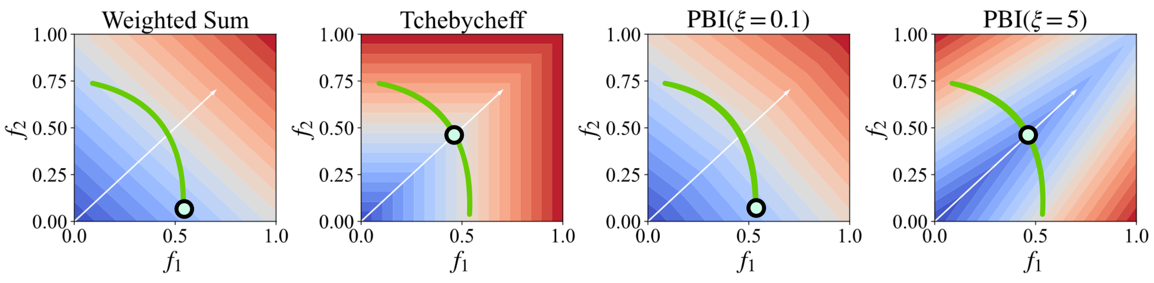


Figure 8: Contour lines of each scalarization function for a two-objective minimization problem. The while arrow represents the request. The green curve and green points represent the Pareto front and the models obtained with different scalarization functions, respectively.

B.2 Results of UniCuCo on Non-Uniform Pruning for Different LLMs

Tables 4 and 5 present additional results for non-uniform pruning. We observe that UniCuCo achieves the same time efficiency in handling compression requests as Uniform, while outperforming Uniform in terms of average accuracy. For instance, UniCuCo outperforms Uniform by approximately 1.6% on Llama-3-8 and by 9% on Llama-2-13B at a sparsity of 70%. Furthermore, although UniCuCo slightly lags behind high-demand time algorithms like OWL and EvoPress in terms of effectiveness, it offers an advantage in terms of time efficiency.

B.3 Impact of Scalarization Function

As mentioned in Section 3.1, our approach optimizes the StratNet using a weighted Tchebycheff scalarization function (Eq. (5)). In this subsection, we compare it with other scalarization functions, such as the weight sum function (Eq. (4)) and the Penalty-based Boundary Intersection (PBI) method. The PBI scalarization function is defined as follows:

$$\min_{\boldsymbol{x}} g_{pbi}(\boldsymbol{x} \mid \boldsymbol{\lambda}) = d_1 + \xi d_2, \qquad (14)$$

where $d_1 = \frac{||(\boldsymbol{z}^* - \boldsymbol{f}(\boldsymbol{x}))^T \boldsymbol{\lambda}||}{||\boldsymbol{\lambda}||}$ and $d_2 = ||\boldsymbol{f} - (\boldsymbol{z}^* - d_1 \boldsymbol{\lambda})||$. $\xi > 0$ represents a penalty parameter. d_1 geometrically represents the distance between $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{\lambda}$, indicating the degree of alignment with the request. d_2 geometrically represents the distance between $\boldsymbol{f}(\boldsymbol{x})$ and the origin when $\boldsymbol{z}^*$ is set to the origin.

To facilitate the analysis of the properties of the above scalarization functions, we present their contour plots for a given white request $\boldsymbol{\lambda}$ in Fig. 8. In these plots, the bluer the color, the smaller the value of the scalarization function, indicating better performance, while the redder the color, the larger the value, indicating poorer performance. Theoretically, the weighted Tchebycheff function can find an optimal solution for any given request, regardless of the shape of the Pareto front. However, the model obtained by weighted sum and PBI ($\xi = 0.1$) cannot accurately align request under concave Pareto fronts. For PBI ($\xi = 5$), the emphasis on aligning requests (due to the large value of ξ) leads to neglecting the scalarization function value, i.e., the distance from the origin. Fig. 9 illustrates the adjusted perplexity (higher values indicate bet-

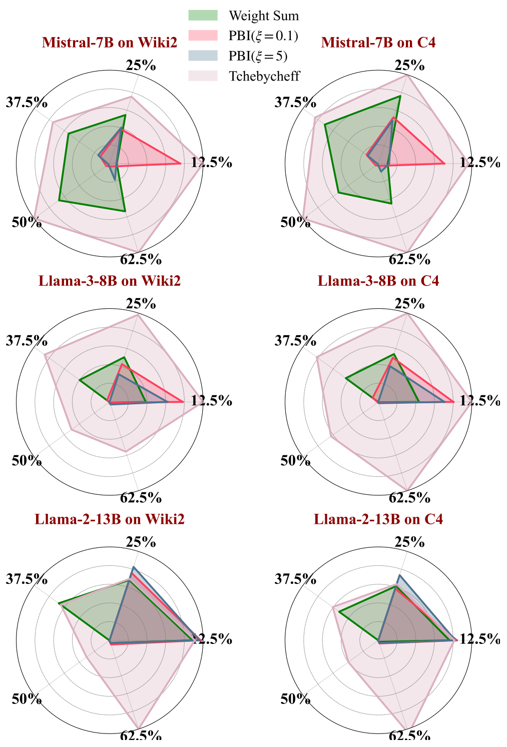


Figure 9: Effects of different scalarization functions on compression performance at five levels of sparsity.

ter performance), where the weighted Tchebycheff function achieves the best results, followed by the Weighted Sum and PBI ($\xi = 0.1$), with PBI ($\xi = 5$) yielding the poorest performance.