Ultra-Efficient and Effective Large Language Models with Multi-Boolean Architectures

Ba-Hien Tran [1] **Van Minh Nguyen** [1]

Abstract

Weight binarization has emerged as a promising strategy to drastically reduce the complexity of large language models (LLMs). It is mainly classified into two approaches: post-training binarization and finetuning with training-aware binarization methods. The first approach, while having low complexity, leads to significant loss of information from the original LLMs, resulting in poor performance. The second approach, on the other hand, relies heavily on full-precision latent weights for gradient approximation of binary weights, which not only remains suboptimal but also introduces substantial complexity. In this paper, we introduce a novel framework that effectively transforms LLMs into multi-kernel Boolean parameters, for the first time, finetunes them directly in the Boolean domain, eliminating the need for expensive latent weights. This significantly reduces complexity during both finetuning and inference. Through extensive and insightful experiments across a wide range of LLMs, we demonstrate that our method outperforms state-of-the-art ultra low-bit quantization and binarization methods.

1. Introduction

Large language models [53, 7, 52, 34] have demonstrated unprecedented capabilities, largely due to the continuous growth in both model and dataset sizes. A key area of focus in optimizing these models is lower-precision computation, which offers substantial benefits in terms of memory and computational efficiency. One prominent approach to achieving this is through the quantization of weight parameters, which reduces the model size by lowering the precision of the weight values. Recent studies on scaling laws [15, 30] have highlighted the potential of using low-precision techniques for large language models (LLMs).

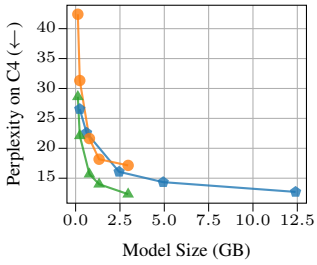


Figure 1: Finetuning OPT models [63] using our 3 Boolean kernels (▲), compared to QPTQ [18] (▲), which quantizes the models to 3 bits, and the FP-16 baseline (⬟) on the C4 dataset.

Binarization represents one of the most extreme forms of quantization for LLMs. While significant progress has been made, challenges remain [61, 23, 32]. Even with advanced techniques like Quantization-Aware Training (QAT), which fine-tunes the model extensively after binarization [58, 25], or trains it from scratch [56], performance still lags behind that of full-precision (FP) models. This performance gap can be attributed to the limited representation capacity of binary weights and the heavy reliance on FP latent weights for binarization. This reliance not only makes the approach computationally expensive but also suboptimal, as it requires gradient approximation. Meanwhile, recent advances in 4-bit quantization have achieved remarkable compression with minimal accuracy loss, but further compression or applying these methods to smaller models has yielded unsatisfactory results [18, 33].

In this paper, we aim to push the boundary of low-precision LLMs by proposing a novel method named as Multiple Boolean Kernels (MBOK). We extend the work in [43], which proposes training neural networks with native Boolean weights directly in the Boolean domain, However, effectively applying this approach to LLMs remains a key challenge. In particular, our contributions are as follows:

- We propose the framework MBOK, which employs multiple Boolean kernels, each using distinct Boolean weights (§ 4.2). This allows for flexibly representing LLMs with low bits, while approaching to FP performance with minimal *both* finetuning and inference cost. The Boolean weights are directly trained in Boolean domain, avoiding the need for FP latent weights and gradient approximations.
- We propose a novel successive method that effectively

[1]Mathematical and Algorithmic Sciences Laboratory, Huawei Paris Research Center, France. Correspondence to: Ba-Hien Tran <ba.hien.tran@huawei.com>.

International Conference on Machine Learning, ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models, Vancouver, Canada. 2025. Copyright 2025 by the author(s).

Table 1: A summary of SOTA binarization methods for LLMs compared to our method.

Method	Train from Scratch	Post-training Binarization	Finetune from FP Model	Calibration Data	Weight Update	Multiple Binary Bases	Higher-bit Salient Weights
BitNet [56]	✓	✗	✗	NA	FP latent-weights	✗	✗
BiLLM [23]	✗	✓	✗	✓	NA	✓	✗
PB-LLM [61]	✗	✓	✗	✓	NA	✗	✓
STBLLM [16]	✗	✓	✗	✓	NA	✓	✓
ARB-LLM [32]	✗	✓	✗	✓	NA	✓	✓
QBB [8]	✗	✓	✓	✓	FP latent-weights	✓	✗
OneBit [58]	✗	✗	✓	✓	FP latent-weights	✗	✗
MoS [25]	✗	✗	✓	✓	FP latent-weights	✗	✗
MBOK [Ours]	✗	✗	✓	✓	Native Boolean weights	✓	✗

transfers knowledge from an FP LLM into the Boolean model (§ 4.3), followed by further fine-tuning using knowledge distillation (§ 4.3.2).

- We introduce a method for automatically allocating the number of kernels for each weight (§ 5).
- We provide a comprehensive empirical analysis and benchmarks, demonstrating our method's superior performance over recent binarization and quantization approaches (see § 6). For example, Fig. 1 shows that our method achieves the best accuracy-compression trade-off, outperforming FP and existing quantization techniques.

2. Related Works

LLMs quantization. Quantization techniques are commonly used to reduce the memory and latency of LLMs. They fall into two categories: QAT, which involves retraining or finetuning in a quantized form, and Post-Training Quantization (PTQ), which can be applied directly without retraining. Due to the difficulty of retraining such large models, most work focuses on PTQ [18, 51, 33, 31], though recent efforts also explore QAT via data-free methods (LLM-QAT [38]), or parameter-efficient fine-tuning like LoRA [13]. A promient PTQ method is QPTQ [18], which introduces one-shot low-bit weight quantization using approximate second-order information. Follow-up work refines this by addressing outliers [27, 14], accounting for activation effects [33, 31], and optimizing quantization parameters (OmniQuant [50]). However, effective quantization is still challenging [59].

Binarization. This represents the most extreme form of quantization, typically using the $\mathbf{sign}(\cdot)$ function with gradients estimated via the straight-through-estimator (STE) [5]. Early work focused on small Transformer models [55] trained or fine-tuned on labeled data [2, 46, 37, 36]. Recent efforts have extended binarization to LLMs. Methods like BiLLM [23], PB-LLM [61], STBLLM [16], and ARB-LLM [32] adopt hybrid PTQ approaches, binarizing non-salient weights while using higher precision for important ones, with calibration data used to adjust scaling factors. QBB [8] further improves this with multiple binary bases and knowledge distillation. In contrast, BitNet [56] replaces linear layers with a custom 1-bit weight structure, BitLinear, and trains the model from scratch. OneBit [58], which decomposes weights into 1-bit components and scaling vectors for QAT, further enhanced by MoS [25] using a mixture of scalings. Despite progress, these methods remain costly due to their dependence on FP latent weights during training. Table 1 summarizes the key characteristics of these methods in comparison to ours.

3. Preliminaries

Notations. We use a standard notation for vectors ($\mathbf{a}$), matrices ($\mathbf{A}$), and scalars (a). The i-th element of a vector $\mathbf{a}$ is $\mathbf{a}_{[i]}$, and the element at the i-th row and j-th column of a matrix $\mathbf{A}$ is $\mathbf{A}_{[i,j]}$. The symbol $\odot$ denotes element-wise multiplication, with broadcasting if needed.

3.1. Neural Network Binarization and the Problems of Full-Precision Latent Weights

Binarization is an effective technique for reducing both the size and computation of deep learning models by converting high-precision weight parameters into 1-bit values [24, 11, 48]. For a linear layer, $\mathbf{Y} = \mathbf{X}\mathbf{W}_{\text{FP}}^\top + \mathbf{b}$, where $\mathbf{X}_{\text{FP}} \in \mathbb{R}^{b\times n}$ is the input data, and $\mathbf{W} \in \mathbb{R}^{m\times n}$ with the input size n and output size m, and $\mathbf{b} \in \mathbb{R}^m$ are the FP weights and bias. Binarization results in $\mathbf{Y} = \alpha \cdot \mathbf{X}\mathbf{W}_{\text{bin}}^\top + \mathbf{b}$, with $\mathbf{W}_{\text{bin}} = \mathbf{sign}(\mathbf{W}_{\text{FP}})$ and α as a scaling factor (e.g., $\alpha = \frac{\|\mathbf{W}_{\text{FP}}\|_1}{m\times n}$) [48].

It is important to note that, during training, the FP weights must be retained for learning the binarized weights. In vanilla gradient descent, binarized weights are updated as $\mathbf{W}_{\text{bin}} = \mathbf{sign}(\mathbf{W}_{\text{FP}} - \eta \cdot \mathbf{G}_{\mathbf{W}_{\text{FP}}})$, where η is the learning rate and $\mathbf{G}_{\mathbf{W}_{\text{FP}}}$ is the gradient of the FP weights. This leads to high memory usage, especially with optimizers like Adam [28], which require storing two additional FP momenta for each parameter. Moreover, the gradient approximation for binarized weights often uses a differentiable proxy, like the STE [5], but this introduces performance drops due to proxy gradient noise. This noise can cause oscillations and instability during training.

3.2. Native Boolean Framework for Neural Networks

To address the issues associated with latent-weight-based approaches, [43, 44] recently proposed a principled framework for directly training Boolean neural networks in the

Boolean domain. Consider the l-th Boolean linear layer; in the forward pass, the output of the next layer is defined as:

$$\mathbf{Y}^{(l)}_{[k,j]} = \mathbf{b}^{(l)}_{[j]} + \sum_{i=1}^{n} \mathrm{L}(\mathbf{X}^{(l)}_{[k,i]}, \mathbf{W}^{(l)}_{[i,j]}), \quad 1 \le j \le m, \quad (1)$$

where k denotes the sample index in the batch, and L is a logic gate such as $\mathbf{and}, \mathbf{or}, \mathbf{xor}$, or $\mathbf{xnor}$; Hereafter, for clarity, we consider $\mathrm{L} = \mathbf{xnor}$ as a concrete example. The weights $\mathbf{W}^{(l)}_{[i,j]}$ are Boolean values $\{\text{TRUE}, \text{FALSE}\}$ or $\{-1, +1\}$, as typically used in practical implementations.

As discussed in [43], the logic gate L can be extended to handle mixed-type data. In this paper, we focus on the case where the input data is real-valued, and the weights are Boolean. Specifically, for an input element $x \in \mathbb{R}$, we define $x_{\text{bool}} = \text{TRUE} \Leftrightarrow x \ge 0$, and $x_{\text{bool}} = \text{FALSE} \Leftrightarrow x < 0$, and $|x|$ its magnitude. The logic operation between a real input $x \in \mathbb{R}$ and a Boolean weight $w \in \mathbb{B}$ is defined according to [43] as follows:

$$\mathbf{xnor}(w, x) \triangleq s, \quad \text{s.t. } s_{\text{bool}} = \mathbf{xnor}(w_{\text{bool}}, x), \quad (2)$$

$$\text{and } |s| = |x|. \quad (3)$$

Backward pass. This layer receives the backpropagation signal from the downstream layer. Specifically, $\mathbf{Z}^{(l)}_{[k,j]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{Y}^{(l)}_{[k,j]}}$ denotes the variation of the loss function $\mathcal{L}$ w.r.t. the output at layer l. To optimize the Boolean weights, we need to compute the corresponding loss signal, denoted as $\mathbf{Q}^{(l)}_{[i,j]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{W}^{(l)}_{[i,j]}}$, which is aggregated over the batch dimension k as:

$$\mathbf{Q}^{(l)}_{[i,j]} = \sum_{k=1}^{b} \mathbf{1}(\mathbf{Q}^{(l)}_{[k,i,j]} = \text{TRUE}) |\mathbf{Q}^{(l)}_{[k,i,j]}| \quad (4)$$

$$- \sum_{k=1}^{b} \mathbf{1}(\mathbf{Q}^{(l)}_{[k,i,j]} = \text{FALSE}) |\mathbf{Q}^{(l)}_{[k,i,j]}|, \quad (5)$$

where $\mathbf{Q}^{(l)}_{[i,j,k]} = \mathbf{xnor}(\mathbf{Z}^{(l)}_{[k,j]}, \mathbf{X}^{(l)}_{[k,i]})$, and $\mathbf{1}(\cdot)$ is the indicator function. The backpropagation signal for the upstream layer, $\mathbf{P}^{(l)}_{[k,j]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{X}^{(l)}_{[k,j]}}$, can be computed in a similar manner.

Boolean optimizer. Given the loss signal, the rule for updating the Boolean weight $\mathbf{W}^{(l)}_{[i,j]}$ to minimize the loss function $\mathcal{L}$ is as $\mathbf{W}^{(l)}_{[i,j]} = \neg \mathbf{W}^{(l)}_{[i,j]}$ if $\mathbf{xnor}(\mathbf{Q}^{(l)}_{[i,j]}, \mathbf{W}^{(l)}_{[i,j]}) = \text{TRUE}$. Based on this update rule, we can develop an optimizer that accumulates the signal $\mathbf{Q}^{(l)}_{[i,j]}$ over training iterations. Specifically, let $\mathbf{W}^{(l),t}_{[i,j]}$ denotes the weight at iteration t, and $\mathbf{M}^{(l),t}_{[i,j]}$ represents its accumulator, initialized as $\mathbf{M}^{(l),0}_{[i,j]} = 0$. The update rule for the accumulator is then defined as:

$$\mathbf{M}^{(l),t+1}_{[i,j]} \leftarrow \beta^t \mathbf{M}^{(l),t}_{[i,j]} + \eta \mathbf{Q}^{(l),t}_{[i,j]}, \quad (6)$$

where η is the accumulation factor acting as a learning rate, and β^t is an auto-regularizing factor that reflects the system's state at time t. In our work, we use brain plasticity [19] and Hebbian theory [21] to adaptively set β^t, as discussed in [43].

Remarks on complexity and applicability to LLMs. This Boolean framework optimizes Boolean parameters $\mathbf{W}^{(l)}_{[i,j]}$ directly in the Boolean space, eliminating the need for FP latent weights. As shown in Eq. 6, the Boolean optimizer is more lightweight than common LLM optimizers like Adam, requiring only one FP momentum per parameter. This reduces both training and inference complexity and avoids gradient approximation induced from STE. As shown in Proposition A.10 in Appendix, $\mathbf{xnor}(w, s) = ws$, mathematically enabling direct application to existing linear algebra operations. Practically, native logic operations are much faster than multiplication.

4. Multiple Boolean Kernels

4.1. Boolean Reformulation for Linear Layers

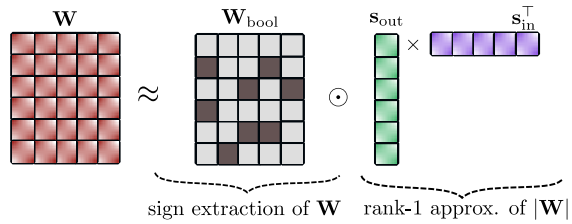


Figure 2: Illustration of SVID.

LLMs [7, 53, 52, 3, 34] are based on the Transformer architecture [55], in which linear layers are the core elements. Inpsired by [58], we employ sign-value-independent decomposition (SVID) such that an FP input matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ of linear layers is decomposed into one Boolean matrix $\mathbf{W}_{\text{bool}} \triangleq \mathbf{sign}(\mathbf{W})$ and two FP vectors $\mathbf{s}_{\text{in}}$ and $\mathbf{s}_{\text{out}}$. Precisely, let $|\mathbf{W}|$ be the element-wise absolute value of $\mathbf{W}$, write $|\mathbf{W}| = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ its singular value decomposition (SVD) [4]. Using rank-1 approximation of $|\mathbf{W}|$, $\mathbf{s}_{\text{in}}$ and $\mathbf{s}_{\text{out}}$ are given as: $\mathbf{s}_{\text{in}} = \sqrt{\sigma_1}\mathbf{V}_{[:,1]}$, and $\mathbf{s}_{\text{out}} = \sqrt{\sigma_1}\mathbf{U}_{[:,1]}$. Then, the input matrix is approximated as $\mathbf{W} = \mathbf{W}_{\text{bool}} \odot |\mathbf{W}| \approx \mathbf{W}_{\text{bool}} \odot \left(\mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right)$. This procedure is illustrated in Fig. 2.

Proposition 4.1 (Xu et al. [58]). *For $\mathbf{W} \in \mathbb{R}^{m \times n}$, write $\mathbf{W} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\widetilde{\mathbf{V}}^\top$ its SVD. Let $\mathbf{a} = \sqrt{\widetilde{\sigma}_1}\widetilde{\mathbf{U}}_{[:,1]}$, and $\mathbf{b} = \sqrt{\widetilde{\sigma}_1}\widetilde{\mathbf{V}}_{[:,1]}$. With the notations as described above, we have:*

$$\left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right\|_F^2 \le \left\|\mathbf{W} - \mathbf{a}\mathbf{b}^\top\right\|_F^2. \quad (7)$$

Remark 4.2. Proposition 4.1 re-states Proposition 2 of [58] with its precise assumption of vectors $\mathbf{a}$ and $\mathbf{b}$ which is necessary for its proof provided in Appendix therein.

Proposition 4.1 shows that using $\mathbf{W}_{\text{bool}}$ together with value matrix approximation is better than a direct rank-1 approximation of $\mathbf{W}$ in terms of Frobenius-norm. This emphasizes the important role of $\mathbf{W}_{\text{bool}}$ in approximating the original FP matrix. Moreover, our following Proposition 4.3 shows that the SVID approximation as described above is optimal for approximating the original matrix $\mathbf{W}_{\text{bool}}$.

Proposition 4.3. *For $\mathbf{W} \in \mathbb{R}^{m\times n}$ and the notations as described above, we have:*

$$\left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right\|_F^2 \leq \left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{c}\mathbf{d}^\top\right\|_F^2, \quad \forall \mathbf{c} \in \mathbb{R}^{m\times 1}, \forall \mathbf{d} \in \mathbb{R}^{n\times 1}. \quad (8)$$

The proof is given in Appendix C.3. The linear layer can be then reformulated as [58]:

$$\mathbf{X}\mathbf{W}_{\text{FP}}^\top \approx \left[\left(\mathbf{X} \odot \mathbf{s}_{\text{in}}^\top\right)\mathbf{W}_{\text{bool}}\right] \odot \mathbf{s}_{\text{out}}^\top. \quad (9)$$

4.2. Enhanced Expressivity with Multiple Kernels

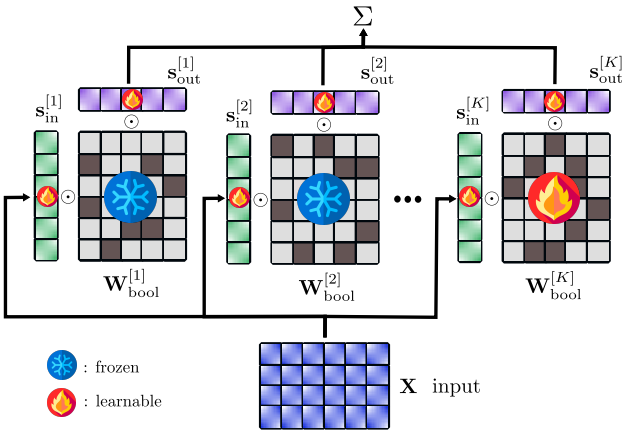


Figure 3: The computation of a linear layer approximated using multi kernels of Boolean.

We have shown that SVID provides a good approximation of the original weights, its expressivity can be still limited to capture well the original FP parameters of complicated models, which were trained on large-scale datasets over extended periods of time. To overcome this limitation, we propose the use of a multi-Boolean kernel structure for the weights. Specifically, we employ K kernels, where each kernel utilizes distinct Boolean weights and scaling factors, to better represent the original weight parameters. This leads to the approximation: $\mathbf{W}_{\text{FP}} \approx \mathbf{W}_{\text{approx}} \triangleq \sum_{k=1}^{K} \mathbf{W}_{\text{bool}}^{[k]} \odot (\mathbf{s}_{\text{out}}^{[k]}\mathbf{s}_{\text{in}}^{[k]\top})$. The computation of a linear layer can then be approximated as follows (see Fig. 3 for an illustration):

$$\mathbf{X}\mathbf{W}_{\text{FP}}^\top \approx \sum_{k=1}^{K}\left[\left(\mathbf{X} \odot \mathbf{s}_{\text{in}}^{[k]\top}\right)\mathbf{W}_{\text{bool}}^{[k]}\right] \odot \mathbf{s}_{\text{out}}^{[k]\top}. \quad (10)$$

Here, the computational costs associated with the FP scaling factors, $\mathbf{s}_{\text{in}}$ and $\mathbf{s}_{\text{out}}$, are small because they only involve element-wise multiplications. The dominant computational cost arises from the matrix multiplication between the scaled input data, $\mathbf{X} \odot \mathbf{s}_{\text{in}}$, and the weights. However, thanks to the use of Boolean weights, the complexity is significantly reduced, as these multiplications can be replaced by additions. Moreover, as we will demonstrate in § 6.1.1, only a small number of kernels are required to achieve a reasonable result. Additionally, we find that, after the successive extraction process from the FP model (§ 4.3.1), we only need to train the Boolean weights for the last kernel and the scaling factors, further significantly reducing the overall complexity.

4.3. Effective Knowledge Transfer into Boolean Models

We have introduced our proposed multi-Boolean kernel structure for effectively representing the linear layers of LLMs. In this section, we outline the process for transferring knowledge from a source FP model to a Boolean model. This process consists of two main steps: (1) an effective data-free initialization step, which maximizes the retention of information from the source FP model, and (2) a data-dependent finetuning step, where the Boolean model is further trained on a target dataset with guidance from the FP model.

4.3.1. Successive Extraction using SVID

For each linear layer, to initialize the values of the Boolean weights and scaling factors for all kernels, we successively apply SVID to the given FP weights. The goal here is to further proceed to SVID process to approximate the residual error introduced by the previous step. Specifically, after each step of decomposing the weight matrix using SVID, we obtain a residual matrix, which is defined as:

$$\mathbf{W}_{\text{res}}^{[k]} = \mathbf{W}_{\text{input}}^{[k]} - \mathbf{W}_{\text{bool}}^{[k]} \odot \left(\mathbf{s}_{\text{out}}^{[k]}\mathbf{s}_{\text{in}}^{[k]\top}\right). \quad (11)$$

Here, $\mathbf{W}_{\text{res}}^{[k]}$ is the residual matrix, and $\mathbf{W}_{\text{bool}}^{[k]}$, $\mathbf{s}_{\text{out}}^{[k]}$ and $\mathbf{s}_{\text{in}}^{[k]}$ are the extracted parameters for the k-th kernel, while $\mathbf{W}_{\text{input}}^{[k]}$ represents the input FP matrix for step k. For the first step, this is the original weight matrix, and for subsequent steps, it is the residual matrix obtained from the previous step.

Fig. 4 illustrate this process. Although using multiple kernels effectively captures the original weight matrix, a residual error still remains at the end of the process. While this residual error is small, it can accumulate as it propagates through the layers, finally leading to predictions that diverge from those of the original FP model. To address this issue, it is necessary to further finetune the resulting model to compensate for these errors and make it better suited to the target task. We will discuss this in § 6.1.2. In the following section, we will introduce knowledge distillation to achieve this goal.

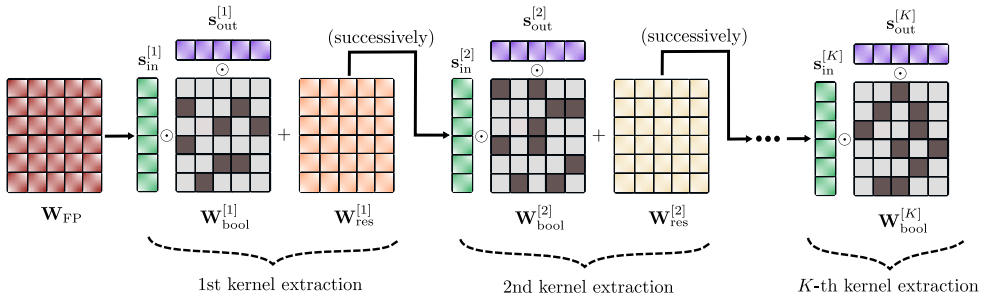


Figure 4: Illustration of successive extractions of Boolean kernels from a given FP weight matrix.

4.3.2. Finetuning with Knowledge Distillation

Knowledge distillation (KD) [22] involves training a student network to replicate the behavior of a teacher network, typically when the student is more efficient. The student is guided by the teacher's output distribution and/or intermediate states as "soft targets". In our case, the FP model serves as the teacher or source model, and the Boolean model is the student. More specifically, the output probability distribution of an LLM for a token $\mathbf{X}_{[i]}$ is computed as:

$$p(\mathbf{X}_{[i]};\tau) = \frac{\exp(\mathbf{X}_{[i]}/\tau)}{\sum_{j=1}^{N_V}\exp(\mathbf{X}_{[j]}/\tau)}, \tag{12}$$

where N_V is the vocabulary size and τ is the softmax temperature. The logit-based knowledge distillation (KD) loss across the sequence of all output tokens is defined as:

$$\mathcal{L}_{\text{logits}} = \frac{1}{L}\sum_{j=1}^{L}\mathrm{D}_{\text{logits}}\left(p_{\text{FP}}(\mathbf{X}_{[j]};\tau), p_{\text{bool}}(\mathbf{X}_{[j]};\tau)\right), \tag{13}$$

where $p_{\text{FP}}(\mathbf{X}_{[j]};\tau)$ and $p_{\text{bool}}(\mathbf{X}_{[j]};\tau)$ denote the distrutions on the j-th token in the input sequence yielded from the FP and Boolean models, respectively, and L is the sequence length. We empirically found that the temperature $\tau = 1$ works best in our experiments. There are several possible choices for the measure $\mathrm{D}_{\text{logits}}$ [29]. We find that the simple forward Kullback–Leibler (KL) divergence yields the best results. A more detailed discussion of this choice is be provided in Appendix E.2.

Additionally, to minimize the distributional discrepancies in the intermedate layers, we incoporate an intermediate state-based KD loss across a squence of hidden states as:

$$\mathcal{L}_{\text{is}} = \frac{1}{L}\sum_{h\in H}\sum_{j=1}^{L}\left\|\mathbf{Q}_{\text{FP}}^{j,h} - \mathbf{Q}_{\text{bool}}^{j,h}\right\|_2^2, \tag{14}$$

where $\mathbf{Q}_{\text{FP}}^{j,h}$ and $\mathbf{Q}_{\text{bool}}^{j,h}$ represent the h-th hidden states of the FP and Boolean models for the j-th token, repsectively; H is the set of chosen intermediate states.

Finally, the overall loss is then computed as $\mathcal{L} = \mathcal{L}_{\text{logits}} + \gamma\mathcal{L}_{\text{is}}$, where γ is a weighted factor that balances the contribution of the two losses. We empirically found that $\gamma = 10$ works best.

5. Kernel Allocation

Using more kernels improves the Boolean model's representation capacity, but it increases the model size. In this section, we present a method for automatically selecting the number of kernels for each weight, given a specific overall budget. Let $N_{\mathbf{W}}$ represent the number of weights in the FP teacher model, and let K_l for $l \in [1, N_{\mathbf{W}}]$ denote the number of target Boolean kernels for the l-th weight. Our goal is to determine $\mathbf{k} \triangleq \{K_l\}_{l\in[1,N_{\mathbf{W}}]}$ while meeting specific design objectives. Several factors must be considered:

(1) Residual error: Denote $e_l^{[k]} \in \mathbb{R}$ the approximation error when apply our successive SVID extraction method to the k-th kernel for the l-th weight, which can be measured using the Frobenius norm of $\mathbf{W}_{\text{res}}^{[k]}$ (Eq. 11).

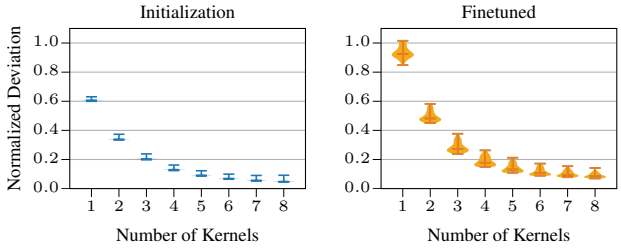
(2) Weight importance: Denote h_l the importance of the l-th weight of the FP teacher model. The importance score is a key factor, as a higher score indicates the need for more Boolean kernels. In this work, we propose using projection weighted canonical correlation analysis (PWCCA) [42] to estimate these scores, as PWCCA is a reliable tool for analyzing representations in deep models. The details of the method to estimate the importance scores are available in Appendix D.1.

(3) Weight size: The size of the l-th weight is denoted by s_l and $p_l \triangleq s_l/\sum_{k=1}^{N_{\mathbf{W}}} s_k$ represents its relative size in the model.

For a given $\mathbf{k}$, the size of the target Boolean model, in terms of the number of weights, is $\sum_{l=1}^{N_{\mathbf{W}}} K_l s_l$. Relative to the source FP model, this repersents an expansion ratio, defined as:

$$\rho(\mathbf{k}) \triangleq \frac{\sum_{l=1}^{N_{\mathbf{W}}} K_l s_l}{\sum_{l=1}^{N_{\mathbf{W}}} s_l} = \sum_{l=1}^{N_{\mathbf{W}}} K_l p_l. \tag{15}$$

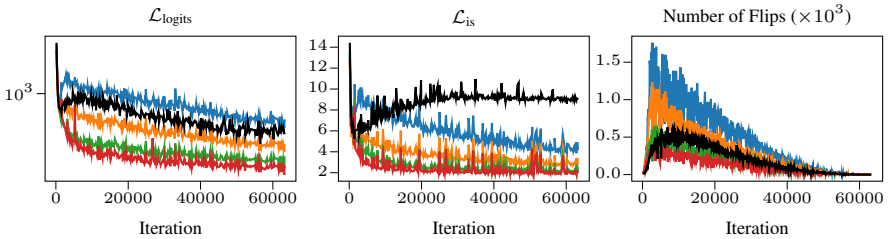
Optimization objective. To control the model size, we constrain the expansion ratio to a target value $T \geq 1$. Additionally, we constrain the kernel size to a upper limit $k_{\max}$, such that $T \leq K_{\max} \leq \infty$. Thus, the optimization space is $\mathcal{K} \triangleq [1, K_{\max}]^{N_{\mathbf{W}}}$. The optimization problem is then formulated as:



Perplexity (↓)

#Kernels	1	2	3	4
Wiki2	39.38	31.47	29.10	28.52
C4	35.44	28.62	26.48	25.90
#Kernels	**5**	**6**	**7**	**8**
Wiki2	28.19	28.16	28.13	28.08
C4	25.65	25.50	25.40	25.31

Figure 5: Normalized L1 norm difference between the approximated weights at initialization and after finetuning against the FP weights ($\|\mathbf{W}_{\text{approx}} - \mathbf{W}_{\text{FP}}\|_1 / \|\mathbf{W}_{\text{FP}}\|_1$) , and the final results.



Optim. Kernel	Wiki2	C4
1st	33.90	30.70
2nd	30.29	27.55
3rd	29.00	26.36
4th	**28.60**	**25.93**
All	32.04	29.08

Figure 6: The progression of training losses, number of flips, and perplexity of the resulting models (OPT-125M) is examined with respect to the optimization of different kernel configurations.

$$\mathbf{k}^* = \arg\min_{\mathbf{k}\in\mathcal{K}} \mathcal{E}(\mathbf{k}), \quad \text{s.t.} \quad \rho(\mathbf{k}) \leq T, \tag{16}$$

$$\text{where } \mathcal{E}(\mathbf{k}) \triangleq \sum_{l=1}^{N_{\mathbf{W}}} h_l e_l^{[K_l]} f(p_l). \tag{17}$$

Here, $\mathcal{E}(\mathbf{k})$ is the objective (energy) function, and $f(\cdot)$ is a certain monotonically decreasing function. Based on our experiments, a practical choice is $f(p_l) = (1/p_l)\log(1/p_l)$. Intuitively, we aims to minimize residual error but also prioritize weights with higher importance and smaller size, balancing the trade-off between accurate knowledge transfer and model efficiency.

Optimization algorithm. The problem's complexity is $\mathcal{O}(K_{\max}^{N_{\mathbf{W}}})$, which can be very large for LLMs. To develop an efficient algorithm for this NP-hard problem, we observe that $e_l^{[k]}$ decreases with k for all l. The energy function $\mathcal{E}(\mathbf{k})$ is maximized when $\mathbf{k} = \mathbf{1}$, and any increase in any component k_l reduces $\mathcal{E}(\mathbf{k})$. This suggests a heuristic iterative approach where the algorithm increments K_l by one unit in the most efficient direction (i.e., determining the best l) with the greatest reduction in $\mathcal{E}(\mathbf{k})$. Our final algorithm is presented in Algorithm 9 in Appendix.

6. Experiments

Setups. In all the experiments, we follow the experimental protocol from [25]. The training dataset consists of a mixed set, combining the WikiText2 [41] training data and a selected partition from the C4 [47] training data, with a sequence length of 2048. We use a cosine decay learning rate scheduler, preceded by a warm-up phase constituting 3% of the total training time, which spans 3 epochs with batch size 8. For Boolean parameters, the maximum learning rate is set to 5×10^3. For the remaniing FP parameters, we employ the AdamW [39] optimizer with a maximum learning rate of 2×10^{-5} and hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$. As a standard evaluation [25, 58], we assess language modeling performance by measuring perplexity (lower is better) on the WikiText2 and C4 datasets.

6.1. Ablation Studies and Analysis

6.1.1. Effect of the Number of Kernels

We begin by examining the effect of the number of Boolean kernels on the OPT-125M model [63]. Fig. 5 shows the normalized difference between the approximated weights using our successive SVID and the original FP weights, both at initialization and after finetuning. As the number of Boolean kernels increases, the approximation error decreases, leading to better perplexity performance. This contrasts with MoS [25], where adding more experts does not consistently improve performance and may even degrade it. This shows the importance role of using additional Boolean weights. As shown in Fig. 5, using 3 or 4 kernels provides a good approximation, with diminishing improvements beyond that. Finally, we note that the normalized difference from the FP weights is even higher after fine-tuning with KD. We hypothesize that KD finetuning compensates the errors due to the lower expressiveness of a small number of kernels, further emphasizing its role in adapting the model to approximate the FP model rather than replicating every individual weight.

6.1.2. Optimization Strategy

Next, we study the effect of optimizing kernels on the OPT-125M model. We consider four Boolean kernels and train

Table 2: Perplexity and zero-shot accuracy results of Float16, quantized and binarized LLMs.

Model	Method	Wbits	Perplexity (↓) Wiki2	C4	Zero-shot Accuracy (↑) BoolQ	PIQA	Hella.	WinoG.	ARC-e	ARC-c	Average
OPT-1.3B [63]	FP-16	16	14.62	14.72	57.82	72.42	53.70	59.51	50.97	29.52	53.99
	PB-LLM [61]	1.7	272.83	175.42	62.17	54.24	27.25	50.27	27.98	23.72	40.94
	BiLLM [23]	1.11	69.45	63.92	61.92	59.52	33.81	49.32	34.38	22.35	43.55
	OneBit [58]	1	20.36	20.76	57.85	66.53	39.21	54.61	42.80	23.97	47.50
	MoS [25]	1	18.45	18.83	60.34	68.66	41.99	53.99	44.87	26.19	49.34
	QPTQ [18]	2	9.5e3	3.8e3	39.60	52.07	25.57	49.33	26.68	23.63	35.15
	LLM-QAT [38]	2	4.9e3	2.1e3	37.83	50.05	25.72	49.72	25.76	25.09	34.07
	OmniQuant [50]	2	42.43	55.64	56.45	60.94	33.39	51.85	38.76	23.38	44.13
	MBOK [Ours]	2×1	**16.13**	**16.61**	58.53	70.67	48.11	56.75	48.19	27.90	**51.69**
LLaMA-7B [53]	FP-16	16	5.68	7.08	73.21	77.42	72.99	66.85	52.53	41.38	64.06
	PB-LLM [61]	1.7	198.37	157.35	60.51	53.53	27.23	49.17	27.48	26.02	40.66
	BiLLM [23]	1.11	41.66	48.15	62.23	58.65	34.64	51.14	33.08	25.68	44.24
	OneBit [58]	1	8.48	10.49	62.50	70.40	54.03	55.32	41.07	30.88	52.36
	MoS [25]	1	7.97	9.72	64.59	71.82	58.18	58.88	42.09	31.31	54.48
	QPTQ [18]	2	1.9e3	7.8e2	43.79	49.95	25.63	49.41	25.84	27.47	37.02
	LLM-QAT [38]	2	7.1e2	3.0e2	37.83	50.87	24.76	51.78	26.26	25.51	36.17
	OmniQuant [50]	2	15.34	26.21	58.69	62.79	43.68	52.96	41.54	29.35	48.17
	MBOK [Ours]	2×1	**6.83**	**8.53**	69.20	74.32	64.80	60.30	49.05	34.90	**58.76**
LLaMA-13B [53]	FP-16	16	5.09	6.61	68.47	79.05	76.24	70.17	59.85	44.54	66.39
	PB-LLM [61]	1.7	35.83	39.79	62.17	58.70	33.97	52.17	31.86	23.63	43.75
	BiLLM [23]	1.11	14.56	16.67	62.53	68.17	52.24	59.43	41.91	29.94	52.37
	OneBit [58]	1	7.65	9.56	63.30	71.98	60.61	59.43	42.85	32.42	55.10
	MoS [25]	1	7.16	8.81	63.82	73.88	64.05	60.93	44.28	33.11	56.68
	QPTQ [18]	2	3.2e3	9.9e2	42.39	50.00	25.27	50.67	26.14	27.39	36.98
	LLM-QAT [38]	2	1.8e3	1.2e3	37.83	50.33	25.40	51.62	27.02	26.87	36.51
	OmniQuant [50]	2	13.43	19.33	62.20	68.99	54.16	53.83	45.50	30.38	52.51
	MBOK [Ours]	2×1	**6.17**	**7.88**	68.10	76.33	69.88	64.17	52.34	37.88	**61.45**

Table 3: OPT perplexity results (*lower is better*) on WikiText2 and C4.

OPT Model	WBits	Wiki2 125M	350M	1.3B	2.7B	6.7B	C4 125M	350M	1.3B	2.7B	6.7B
FULL-PRECISION	16	27.65	22.00	14.63	12.47	10.86	26.56	22.59	16.07	14.34	12.71
RTN [60, 12]	3	37.28	25.94	48.17	16.92	12.10	33.91	26.21	24.51	18.43	14.36
QPTQ [18]	3	31.12	24.24	15.47	12.87	11.39	29.22	24.63	16.97	15.00	13.18
MBOK [Ours]	3×1	**29.10**	**23.12**	**15.30**	**13.09**	**11.03**	**28.62**	**22.10**	**15.68**	**14.00**	**12.33**

only one, freezing the others. Fig. 6 shows the convergence of the losses. Training the first kernel results in the slowest convergence, with progressively better performance as higher-order kernels are trained. As shown in Proposition 4.1 and Proposition 4.3, the SVID effectively extracts optimal Boolean weights and scaling factors. In our successive SVID framework, the first kernel is well extracted and captures the most important information, while higher-order kernels approximate the residuals. The kernels are related in a successive manner, so modifying the lower-order kernels affects the higher-order kernels. We observe that the number of flips of the Boolean weights is very high when training only the first kernel, indicating that the model struggles to optimize. In contrast, fine-tuning only the last kernel to compensate for final approximation errors and adapt to the data is more efficient, as reflected by the lowest flip rates and best performance. This is in line with the observation in [35], where they compress "delta" induced by the finetuning process by using 1-bit weights. Additionally, training all the kernels together performs worse due to interference between them. This further highlights the advantage of our approach, as training complexity is significantly reduced by only optimizing the last kernel. Thus, we apply this strategy in all our experiments.

6.2. Main Benchmark Results

Table 2 compares our method with recent baselines in binarization and 2-bit quantization, evaluating perplexity and accuracy on zero-shot tasks including Winogrande [49], HellaSwag [62], PIQA [6], BoolQ [9], and ARC [10]. For our method, we use 2 Boolean kernels, an ultra low-bit setting.

Due to space constraints, the results for LLaMA2-7B and LLaMA2-13B [54] and different number of Boolean kernels are provided in Appendix E.4 and Appendix E.3.

Our method consistently and significantly outperforms the baselines in both perplexity and zero-shot accuracy, achieving results close to the FP-16 baseline despite using only a budget of 2 bits for weight. As expected, QAT methods like OneBit and MoS perform better than PTQ methods, but this comes at the cost of extensive finetuning. In contrast, our approach efficiently address this problem by optimizing parameters directly in Boolean space, avoiding the need for optimizing in FP latent sapce.

6.3. Accuracy-Compression Trade-offs

We further investigate the accuracy-compression trade-offs of our method, quantization methods, and the FP model. Specifically, we compare 3-bit quantization using round-to-nearest (RTN) [60, 12] and QPTQ [18] methods against our approach using 3 Boolean kernels. We evaluate these methods on OPT models of varying sizes. The results, presented in Table 3 and Fig. 1, show that with 3 kernels, our method closely approaches the performance of the FP model. Given the same weight budget, our method clearly sits on the Pareto frontier, delivering the best performance for the same model size.

6.4. Kernel Allocation and Comparison to BitNet b1.58

We evaluate the effectiveness of our kernel allocation algorithm on the OPT-125M model by comparing a fixed number of kernels (e.g., 2, 3, 4) applied to all weights against an optimized number for each weight. As shown in Fig. 7, the fixed approach performs slightly better for low bit-budget and similarly at high bit-budget. We hypothesize that in extreme cases of small bit-budget, the low-order kernel is crucial for preserving knowledge from the source FP model, as discussed in § 6.1.2. Removing any of these kernels negatively impacts performance. Thus, we recommend using a fixed number of kernels in such cases to maintain network coherence.

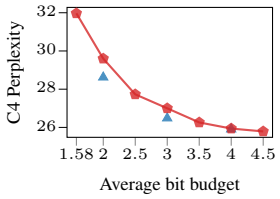


Figure 7: Comparison between using optimized (⬟) and fixed (▲) number of kernels for OPT-125M.

Moreover, our kernel allocation method is particularly beneficial when the target expansion (bit-budget) is arbitrary, rather than an integer. Users can specify any budget for the model depending on the context. For example, this flexibility allows direct comparison with BitNet-b1.58 [40], which uses ternary weights. Our optimized model with a bit-budget of 1.58 achieves reasonable results, while the C4 perplexity of BitNet-b1.58 is 10199.89, due to instability during finetuning, as also noted in [58].

6.5. Discussion on Finetuning Complexity

We highlight the efficiency of our method during finetuning. We compare MoS [25] with our method, using 3 Boolean kernels on OPT-6.7B model. Since we optimize directly in the Boolean domain, each Boolean weight requires only 1 bit, while MoS uses 16 bits for latent weights. Additionally, we only need to finetune the last Boolean kernel, and the Boolean optimizer stores a single 16-bit momentum for each weight. In contrast, the Adam optimizer [28] for latent weights requires two 16-bit momenta per weight. Fig. 8 illustrates the estimated memory for a minibatch size of one, clearly demonstrating the significant memory savings of our method compared to MoS during finetuning. The memory gain can be even more significant if we incorporate recent advancements in compressing optimizer states, such as GaLore [64].

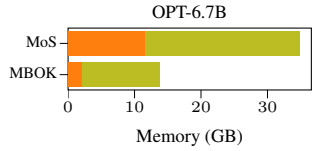


Figure 8: Estimated memory for finetuning for weights (■) and optimizer states (■).

7. Conclusions

We introduced Multiple Boolean Kernels (MBOK), a novel framework for low-bit finetuning LLMs. By utilizing Boolean weights and optimizing them directly in the Boolean domain, our framework significantly reduces both memory and computation costs during *both* finetuning and inference. The flexible multi-Boolean structure, along with the proposed successive SVID, effectively transfers knowledge from a source FP model. Through extensive experiments on LLMs of various sizes, we demonstrate that our method approaches FP performance while achieving the best accuracy-compression trade-off compared to existing quantization and binarization methods.

Limitations. Our method, like other binarized neural networks, could not be assessed on native Boolean accelerators due to hardware being optimized for real arithmetic. However, it may inspire the development of hardware tailored for Boolean processing. Additionally, while our kernel allocation algorithm is effective in some cases, it does not show significant improvement over using a fixed number of kernels for all weights. We stress that this is an NP-hard problem, and our algorithm only considers the initialization stage, meaning the optimized solution can be heavily influenced by the finetuning process. Future work should focus on improving the optimization algorithm and incorporating finetuning dynamics to better select the number of kernels. Lastly, we did not consider activation or gradient quantization, which could further reduce memory and computation costs and is left for future exploration.

References

[1] R. Agarwal, N. Vieillard, Y. Zhou, P. Stanczyk, S. R. Garea, M. Geist, and O. Bachem. On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes. In *The Twelfth International Conference on Learning Representations*, 2024.

[2] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King. BinaryBERT: Pushing the Limit of BERT Quantization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4334–4348, Online, 2021. Association for Computational Linguistics.

[3] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, et al. Qwen Technical Report. *arXiv preprint arXiv:2309.16609*, 2023.

[4] E. Beltrami. Sulle funzioni bilineari, giomale di mathematiche ad uso studenti delle uninersita. 11, 98–106.(an english translation by d boley is available as university of minnesota, department of computer science). Technical report, Technical Report 90–37, 1990.

[5] Y. Bengio, N. Léonard, and A. Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432*, 2013.

[6] Y. Bisk, R. Zellers, J. Gao, Y. Choi, et al. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[8] A. Bulat, Y. Ouali, and G. Tzimiropoulos. QBB: Quantization with Binary Bases for LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[9] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[10] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[11] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[12] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In *Advances in Neural Information Processing Systems*, 2022.

[13] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. Curran Associates, Inc., 2023.

[14] T. Dettmers, R. A. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler, and D. Alistarh. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. In *The Twelfth International Conference on Learning Representations*, 2024.

[15] T. Dettmers and L. Zettlemoyer. The case for 4-bit precision: k-bit Inference Scaling Laws. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 7750–7774. PMLR, 23–29 Jul 2023.

[16] P. Dong, L. Li, Y. Zhong, D. Du, R. FAN, Y. Chen, Z. Tang, Q. Wang, W. Xue, Y. Guo, and X. Chu. STBLLM: Breaking the 1-Bit Barrier with Structured Binary LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025.

[17] C. Eckart and G. Young. The Approximation of One Matrix by Another of Lower Rank. *Psychometrika*, 1936.

[18] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

[19] E. Fuchs, G. Flügge, et al. Adult Neuroplasticity: More than 40 Years of Research. *Neural plasticity*, 2014, 2014.

[20] A. Gromov, K. Tirumala, H. Shapourian, P. Glorioso, and D. Roberts. The Unreasonable Ineffectiveness of the Deeper Layers. In *The Thirteenth International Conference on Learning Representations*, 2025.

[21] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Psychology press, 2005.

[22] G. Hinton, V. Oriol, and J. Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 1, 2015.

[23] W. Huang, Y. Liu, H. Qin, Y. Li, S. Zhang, X. Liu, M. Magno, and X. Qi. BiLLM: Pushing the Limit of Post-Training Quantization for LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 20023–20042. PMLR, 21–27 Jul 2024.

[24] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[25] D. Jo, T. Kim, Y. Kim, and J.-J. Kim. Mixture of Scales: Memory-Efficient Token-Adaptive Binarization for Large Language Models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[26] C. Jordan. *Calculus of Finite Differences*. Chelsea Publishing Company, New York, 2nd edition, 1950.

[27] S. Kim, C. R. C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer. SqueezeLLM: Dense-and-Sparse Quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 23901–23923. PMLR, 21–27 Jul 2024.

[28] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[29] J. Ko, S. Kim, T. Chen, and S.-Y. Yun. DistiLLM: Towards Streamlined Distillation for Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 24872–24895. PMLR, 21–27 Jul 2024.

[30] T. Kumar, Z. Ankner, B. F. Spector, B. Bordelon, N. Muennighoff, M. Paul, C. Pehlevan, C. Re, and A. Raghunathan. Scaling Laws for Precision. In *The Thirteenth International Conference on Learning Representations*, 2025.

[31] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park. OWQ: Outlier-aware Weight Quantization for Efficient Fine-tuning and Inference of Large Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13355–13364, 2024.

[32] Z. Li, X. Yan, T. Zhang, H. Qin, D. Xie, J. Tian, Z. Shi, L. Kong, Y. Zhang, and X. Yang. ARB-LLM: Alternating Refined Binarizations for Large Language Models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[33] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems*, volume 6, pages 87–100, 2024.

[34] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, et al. DeepSeek-V3 Technical Report. *arXiv preprint arXiv:2412.19437*, 2024.

[35] J. Liu, G. Xiao, K. Li, J. D. Lee, S. Han, T. Dao, and T. Cai. BitDelta: Your Fine-Tune May Only Be Worth One Bit. In *Advances in Neural Information Processing Systems*, volume 37, pages 13579–13600. Curran Associates, Inc., 2024.

[36] Z. Liu, B. Oguz, A. Pappu, Y. Shi, and R. Krishnamoorthi. Binary and Ternary Natural Language Generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 65–77, Toronto, Canada, 2023. Association for Computational Linguistics.

[37] Z. Liu, B. Oguz, A. Pappu, L. Xiao, S. Yih, M. Li, R. Krishnamoorthi, and Y. Mehdad. BiT: Robustly Binarized Multi-distilled Transformer. In *Advances in Neural Information Processing Systems*, volume 35, pages 14303–14316. Curran Associates, Inc., 2022.

[38] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, and V. Chandra. LLM-QAT: Data-Free Quantization Aware Training for Large Language Models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 467–484, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[39] I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019.

[40] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei. The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits. *arXiv preprint arXiv:2402.17764*, 1, 2024.

[41] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations*, 2017.

[42] A. Morcos, M. Raghu, and S. Bengio. Insights on Representational Similarity in Neural Networks with Canonical Correlation. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[43] V. M. Nguyen, C. Ocampo, A. Askri, L. Leconte, and B.-H. Tran. BOLD: Boolean Logic Deep Learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[44] V. M. Nguyen, C. Ocampo, A. Askri, and B.-H. Tran. Boolean Logic for Low-Energy Deep Learning. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*, 2024.

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[46] H. Qin, Y. Ding, M. Zhang, Q. YAN, A. Liu, Q. Dang, Z. Liu, and X. Liu. BiBERT: Accurate Fully Binarized BERT. In *International Conference on Learning Representations*, 2022.

[47] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[48] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, October 2016.

[49] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: An Adversarial Winograd Schema Challenge at Scale. *Communications of the ACM*, 64(9):99–106, 2021.

[50] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. In *The Twelfth International Conference on Learning Representations*, 2024.

[51] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Re, I. Stoica, and C. Zhang. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 31094–31116. PMLR, 23–29 Jul 2023.

[52] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, et al. Gemma: Open Models Based on Gemini Research and Technology. *arXiv preprint arXiv:2403.08295*, 2024.

[53] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*, 2023.

[54] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288*, 2023.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[56] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei. BitNet: Scaling 1-bit Transformers for Large Language Models. *arXiv preprint arXiv:2310.11453*, 2023.

[57] Y. Wen, Z. Li, W. Du, and L. Mou. f-Divergence Minimization for Sequence-Level Knowledge Distillation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10817–10834, Toronto, Canada, 2023. Association for Computational Linguistics.

[58] Y. Xu, X. Han, Z. Yang, S. Wang, Q. Zhu, Z. Liu, W. Liu, and W. Che. OneBit: Towards Extremely Low-bit Large Language Models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[59] Z. Xu, S. Sharify, W. Yazar, T. J. Webb, and X. Wang. Understanding the Difficulty of Low-Precision Post-Training Quantization for LLMs. In *ICLR 2025 Workshop on Sparsity in LLMs*, 2025.

[60] Z. Yao, R. Y. Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. In *Advances in Neural Information Processing Systems*, 2022.

[61] Z. Yuan, Y. Shang, and Z. Dong. PB-LLM: Partially Binarized Large Language Models. In *The Twelfth International Conference on Learning Representations*, 2024.

[62] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics.

[63] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. OPT: Open Pre-trained Transformer Language Models. *arXiv preprint arXiv:2205.01068*, 2022.

[64] J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, and Y. Tian. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 61121–61143. PMLR, 21–27 Jul 2024.

Appendix

A. Primer on Boolean Neural Networks

For completeness, this section reviews the concepts and methodology of Boolean neural networks as proposed by [43].

A.1. Neuron Design

Boolean Neuron. Consider the l-th Boolean linear layer; in the forward pass, the output of the next layer is defined as [43]:

$$\mathbf{Y}^{(l)}_{[k,j]} = \mathbf{b}^{(l)}_{[j]} + \sum_{i=1}^{n} \mathrm{L}(\mathbf{X}^{(l)}_{[k,i]}, \mathbf{W}^{(l)}_{[i,j]}), \qquad 1 \leq j \leq m, \tag{18}$$

where k denotes the sample index in the batch, and L is a logic gate such as $\mathbf{and}, \mathbf{or}, \mathbf{xor}$, or $\mathbf{xnor}$; The weights $\mathbf{W}^{(l)}_{[i,j]}$ are Boolean values $\{\text{TRUE}, \text{FALSE}\}$ or $\{-1, +1\}$, as typically used in practical implementations. n and m are the number of input and output neurons, respectively. As the most extreme use case, the input data are also Boolean values. The above summation is understood as the counting of TRUE values. We emphasize that the framework is flexible, as it allows Boolean linear layers to be connected through activation layers, layer normalization, arithmetic layers, or other types of layers.

Mixed Boolean-Real Neuron. To enable flexible integration and coexistence of Boolean designs with real-valued components in deep models, we consider two cases of mixed-type data: (i) Boolean weights with real-valued inputs, and (ii) real-valued weights with Boolean inputs. This paper focuses on the first case. These scenarios are addressed through an extension of Boolean logic to accommodate mixed-type data. To proceed, we introduce the essential notations and definitions. Specifically, we define $\mathbb{B} \triangleq \{\text{TRUE}, \text{FALSE}\}$ as the Boolean domain, equipped with standard Boolean logic operations.

***Definition* A.1** (Three-valued logic)**.** *We define the mixed logic domain as $\mathbb{M} \triangleq \mathbb{B} \cup \{0\}$, where 0 represents an undefined or neutral value. The logic connectives in $\mathbb{M}$ are defined in alignment with standard Boolean logic, as follows. First, the negation operator is extended as: $\neg\text{TRUE} = \text{FALSE}$, $\neg\text{FALSE} = \text{TRUE}$, and $\neg 0 = 0$. Next, let L denote a generic logic connective (e.g.,* AND, OR*). We distinguish its use in $\mathbb{M}$ and $\mathbb{B}$ by writing $\mathrm{L}_{\mathbb{M}}$ and $\mathrm{L}_{\mathbb{B}}$, respectively. The extended connective $\mathrm{L}_{\mathbb{M}}$ is defined by:*

$$\mathrm{L}_{\mathbb{M}}(a, b) = \begin{cases} \mathrm{L}_{\mathbb{B}}(a, b) & \textit{for } a, b \in \mathbb{B}, \\ 0 & \textit{otherwise}. \end{cases}$$

Notation A.2. Denote by $\mathbb{L}$ a logic set (e.g., $\mathbb{B}$ or $\mathbb{M}$), $\mathbb{R}$ the real set, $\mathbb{Z}$ the set of integers, $\mathbb{N}$ a numeric set (e.g., $\mathbb{R}$ or $\mathbb{Z}$), and $\mathbb{D}$ a certain set of $\mathbb{L}$ or $\mathbb{N}$.

***Definition* A.3.** *For $x \in \mathbb{N}$, its logic value denoted by x_{logic} is given as $x_{\text{logic}} = \text{TRUE} \Leftrightarrow x > 0$, $x_{\text{logic}} = \text{FALSE} \Leftrightarrow x < 0$, and $x_{\text{logic}} = 0 \Leftrightarrow x = 0$.*

***Definition* A.4.** *The magnitude of a variable x, denoted by $|x|$, is defined as follows. If $x \in \mathbb{N}$, then $|x|$ is the standard absolute value. For $x \in \mathbb{L}$, the magnitude is given by:*

$$|x| = \begin{cases} 0 & \textit{if } x = 0, \\ 1 & \textit{otherwise}. \end{cases}$$

***Definition* A.5** (Mixed-type logic)**.** *For L a logic connective of $\mathbb{L}$ and variables a, b, operation $c = \mathrm{L}(a, b)$ is defined such that $|c| = |a||b|$ and $c_{\text{logic}} = \mathrm{L}(a_{\text{logic}}, b_{\text{logic}})$.*

A.2. Mathematical Foundations of Boolean Variation

In this section, we present the mathematical foundation of Boolean variation which is the corner stone of the method for training Boolean weights directly within the Boolean domain, without relying on FP latent weights [43].

A.2.1. Boolean Variation

***Definition* A.6.** *Order relations '$<$' and '$>$' in $\mathbb{B}$ are defined as follows:*

$$\text{FALSE} < \text{TRUE}, \quad \text{TRUE} > \text{FALSE}. \tag{19}$$

***Definition* A.7.** *For $a, b \in \mathbb{B}$, the variation from a to b, denoted $\delta(a \to b)$, is defined as:*

$$\delta(a \to b) \triangleq \begin{cases} \text{TRUE}, & \text{if } b > a, \\ 0, & \text{if } b = a, \\ \text{FALSE}, & \text{if } b < a. \end{cases} \tag{20}$$

***Definition* A.8** (Type conversion)**.** *Define:*

$$\begin{aligned} \mathrm{p}\colon \mathbb{N} &\to \mathbb{L} \\ x &\mapsto \mathrm{p}(x) = \begin{cases} \text{TRUE}, & \text{if } x > 0, \\ 0, & \text{if } x = 0, \\ \text{FALSE}, & \text{if } x < 0. \end{cases} \end{aligned} \tag{21}$$

Proposition A.9 (Nguyen et al. [43])**.** *The following properties hold:*

1. $\forall x, y \in \mathbb{N}$: $\mathrm{p}(xy) = \mathbf{xnor}(\mathrm{p}(x), \mathrm{p}(y))$.
2. $\forall a, b \in \mathbb{L}$: $\mathrm{e}(\mathbf{xnor}(a, b)) = \mathrm{e}(a)\,\mathrm{e}(b)$.
3. $\forall x, y \in \mathbb{N}$: $x = y \Leftrightarrow |x| = |y|$ *and* $\mathrm{p}(x) = \mathrm{p}(y)$.

In particular, property Proposition A.9(2) implies that by the embedding map $\mathrm{e}(\cdot)$, we have:

$$(\{\text{TRUE}, \text{FALSE}\}, \mathbf{xor}) \cong (\{\pm 1\}, -\times), \tag{22}$$

$$(\{\text{TRUE}, \text{FALSE}\}, \mathbf{xnor}) \cong (\{\pm 1\}, \times), \tag{23}$$

where $\cong$ and $\times$ stand for isomorphic relation, and the real multiplication, resp. A consequence is that by $\mathrm{e}(\cdot)$, a computing sequence of pointwise XOR or XNOR, counting, and majority vote is equivalent to a sequence of pointwise multiplications and accumulation performed on the embedded data.

Proposition A.10. *The following properties hold:*

1. $a \in \mathbb{L}$, $x \in \mathbb{N}$: $\mathbf{xnor}(a, x) = \mathrm{e}(a)x$.
2. $x, y \in \mathbb{N}$: $\mathbf{xnor}(x, y) = xy$.
3. $x \in \{\mathbb{L}, \mathbb{N}\}$, $y, z \in \mathbb{N}$: $\mathbf{xnor}(x, y + z) = \mathbf{xnor}(x, y) + \mathbf{xnor}(x, z)$.
4. $x \in \{\mathbb{L}, \mathbb{N}\}$, $y, \lambda \in \mathbb{N}$: $\mathbf{xnor}(x, \lambda y) = \lambda \mathbf{xnor}(x, y)$.
5. $x \in \{\mathbb{L}, \mathbb{N}\}$, $y \in \mathbb{N}$: $\mathbf{xor}(x, y) = -\mathbf{xnor}(x, y)$.

Proof. The proof follows definitions A.5 and A.8.

- Following Definition A.1 we have $\forall t \in \mathbb{M}$, $\mathbf{xnor}(\text{TRUE}, t) = t$, $\mathbf{xnor}(\text{FALSE}, t) = \neg t$, and $\mathbf{xnor}(0, t) = 0$. Put $v = \mathbf{xnor}(a, x)$. We have $|v| = |x|$ and $\mathrm{p}(v) = \mathbf{xnor}(a, \mathrm{p}(x))$. Hence, $a = 0 \Rightarrow \mathrm{p}(v) = 0 \Rightarrow v = 0$; $a = \text{TRUE} \Rightarrow \mathrm{p}(v) = \mathrm{p}(x) \Rightarrow v = x$; $a = \text{FALSE} \Rightarrow \mathrm{p}(v) = \neg\,\mathrm{p}(x) \Rightarrow v = -x$. Hence (1).
- The result is trivial if $x = 0$ or $y = 0$. For $x, y \neq 0$, put $v = \mathbf{xnor}(x, y)$, we have $|v| = |x||y|$ and $\mathrm{p}(v) = \mathbf{xnor}(\mathrm{p}(x), \mathrm{p}(y))$. According to Definition A.8, if $\mathrm{sign}(x) = \mathrm{sign}(y)$, we have $\mathrm{p}(v) = \text{TRUE} \Rightarrow v = |x||y| = xy$. Otherwise, i.e., $\mathrm{sign}(x) = -\,\mathrm{sign}(y)$, $\mathrm{p}(v) = \text{FALSE} \Rightarrow v = -|x||y| = xy$. Hence (2).
- (3) and (4) follow (1) for $x \in \mathbb{L}$ and follow (2) for $x \in \mathbb{N}$.
- For (5), write $u = \mathbf{xor}(x, y)$ and $v = \mathbf{xnor}(x, y)$, we have $|u| = |v|$ and $\mathrm{p}(u) = \mathbf{xor}(\mathrm{p}(x), \mathrm{p}(y)) = \neg\mathbf{xnor}(\mathrm{p}(x), \mathrm{p}(y)) = \neg\,\mathrm{p}(v)$. Thus, $\mathrm{sign}(u) = -\,\mathrm{sign}(v) \Rightarrow u = -v$. □

Notation A.11. We denote $\mathcal{F}(\mathbb{S}, \mathbb{T})$ the set of all functions from source $\mathbb{S}$ to image $\mathbb{T}$.

***Definition* A.12.** *For $f \in \mathcal{F}(\mathbb{B}, \mathbb{D})$, $\forall x \in \mathbb{B}$, write $\delta f(x \to \neg x) := \delta(f(x) \to f(\neg x))$. The variation of f w.r.t. x, denoted $f'(x)$, is defined as:*

$$f'(x) \triangleq \mathbf{xnor}(\delta(x \to \neg x), \delta f(x \to \neg x)).$$

Remark A.13. For convenience and consistency of notation, we intentionally adopt the standard symbol for the continuous derivative, f', to also denote Boolean variation The intended meaning — whether it represents a continuous derivative or a Boolean variation — can be inferred from the context in which the function f is defined. Intuitively, the variation of f w.r.t x is TRUE if f varies in the same direction with x.

Example A.14. Let $a \in \mathbb{B}$, $f(x) = \mathbf{xor}(x, a)$ for $x \in \mathbb{B}$, the variation of f w.r.t. x can be derived by establishing a truth table (see Table 4) from which we obtain $f'(x) = \neg a$.

Table 4: Variation truth table of $f(x) = \mathbf{xor}(a, x)$, $a, x \in \mathbb{B}$.

a	x	$\neg x$	$\delta(x \to \neg x)$	$f(a, x)$	$f(a, \neg x)$	$\delta f(x \to \neg x)$	$f'(x)$
TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE

A.2.2. Boolean Variation Calculus

Below are some rules of Boolean variation which are necessary for training Boolean neural networks.

Proposition A.15 (Nguyen et al. [43])**.** *For $f, g \in \mathcal{F}(\mathbb{B}, \mathbb{B})$, $\forall x, y \in \mathbb{B}$ the following properties hold:*

1. $\delta f(x \to y) = \mathbf{xnor}(\delta(x \to y), f'(x))$.
2. $(\neg f(x))' = \neg f'(x)$.
3. $(g \circ f)'(x) = \mathbf{xnor}(g'(f(x)), f'(x))$.

Proof. The proof is by definition:

1. $\forall x, y \in \mathbb{B}$, there are two cases. If $y = x$, then the result is trivial. Otherwise, i.e., $y = \neg x$, by definition we have:

$$\begin{aligned} & f'(x) = \mathbf{xnor}(\delta(x \to \neg x), \delta f(x \to \neg x)) \\ \Leftrightarrow \quad & \delta f(x \to \neg x) = \mathbf{xnor}(\delta(x \to \neg x), f'(x)). \end{aligned}$$

   Hence the result.

2. $\forall x, y \in \mathbb{B}$, it is easy to verify by truth table that $\delta(\neg f(x \to y)) = \neg \delta f(x \to y)$. Hence, by definition,

$$\begin{aligned}(\neg f)'(x) &= \mathbf{xnor}(\delta(x \to \neg x), \delta(\neg f(x \to \neg x))) \\ &= \mathbf{xnor}(\delta(x \to \neg x), \neg \delta f(x \to \neg x)) \\ &= \neg \mathbf{xnor}(\delta(x \to \neg x), \delta f(x \to \neg x)) \\ &= \neg f'(x).\end{aligned}$$

3. Using definition, property (i), and associativity of $\mathbf{xnor}$, $\forall x \in \mathbb{B}$ we have:

$$\begin{aligned}(g \circ f)'(x) &= \mathbf{xnor}(\delta(x \to \neg x), \delta g(f(x) \to f(\neg x))) \\ &= \mathbf{xnor}(\delta(x \to \neg x), \mathbf{xnor}(\delta f(x \to \neg x), g'(f(x)))) \\ &= \mathbf{xnor}(g'(f(x)), \mathbf{xnor}(\delta(x \to \neg x), \delta f(x \to \neg x))) \\ &= \mathbf{xnor}(g'(f(x)), f'(x)).\end{aligned}$$

□

Proposition A.16 (Nguyen et al. [43]). *For $f \in \mathcal{F}(\mathbb{B}, \mathbb{N})$, the following properties hold:*

1. *$x, y \in \mathbb{B}$: $\delta f(x \to y) = \mathbf{xnor}(\delta(x \to y), f'(x))$.*
2. *$\alpha \in \mathbb{N}$: $(\alpha f)'(x) = \alpha f'(x)$.*
3. *$g \in \mathcal{F}(\mathbb{B}, \mathbb{N})$: $(f + g)'(x) = f'(x) + g'(x)$.*

Proof. The proof is as follows:

1. For $x, y \in \mathbb{B}$. Firstly, the result is trivial if $y = x$. For $y \neq x$, i.e., $y = \neg x$, by definition:

$$f'(x) = \mathbf{xnor}(\delta(x \to \neg x), \delta f(x \to \neg x)).$$

Hence, $|\delta f(x \to \neg x)| = |f'(x)|$ since $|\delta(x \to \neg x)| = 1$, and

$$\begin{aligned} & \mathrm{p}(f'(x)) = \mathbf{xnor}(\delta(x \to \neg x), \mathrm{p}(\delta f(x \to \neg x))) \\ \Leftrightarrow \quad & \mathrm{p}(\delta f(x \to \neg x)) = \mathbf{xnor}(\delta(x \to \neg x), \mathrm{p}(f'(x))),\end{aligned}$$

where $\mathrm{p}(\cdot)$ is the logic projector Eq. 21. Thus, $\delta f(x \to \neg x) = \mathbf{xnor}(\delta(x \to \neg x), f'(x))$. Hence the result.

2. Firstly $\forall x, y \in \mathbb{B}$, we have

$$\delta(\alpha f(x \to y)) = \alpha f(y) - \alpha f(x) = \alpha \delta f(x \to y).$$

Hence, by definition,

$$\begin{aligned}(\alpha f)'(x) &= \mathbf{xnor}(\delta(x \to \neg x), \delta(\alpha f(x \to \neg x))) \\ &= \mathbf{xnor}(\delta(x \to \neg x), \alpha \delta f(x \to \neg x)) \\ &= \alpha\, \mathbf{xnor}(\delta(x \to \neg x), \delta f(x \to \neg x)), \text{ due to Proposition A.10(4)} \\ &= \alpha f'(x).\end{aligned}$$

3. For $f, g \in \mathcal{F}(\mathbb{B}, \mathbb{N})$,

$$\begin{aligned}(f + g)'(x) &= \mathbf{xnor}(\delta(x \to \neg x), \delta(f + g)(x \to \neg x)) \\ &= \mathbf{xnor}(\delta(x \to \neg x), \delta f(x \to \neg x) + \delta g(x \to \neg x)) \\ &\overset{(*)}{=} \mathbf{xnor}(\delta(x \to \neg x), \delta f(x \to \neg x)) + \mathbf{xnor}(\delta(x \to \neg x), \delta g(x \to \neg x)), \\ &= f'(x) + g'(x),\end{aligned}$$

where $(*)$ is due to Proposition A.10(3). □

For $f \in \mathcal{F}(\mathbb{Z}, \mathbb{N})$, its derivative, also known in terms of *finite differences*, has been defined in the literature as $f'(x) = f(x+1) - f(x)$, see e.g. [26]. With the logic variation as introduced above, we can make this definition more generic as follows.

***Definition* A.17.** *For $f \in \mathcal{F}(\mathbb{Z}, \mathbb{D})$, the variation of f w.r.t $x \in \mathbb{Z}$ is defined as $f'(x) \triangleq \delta f(x \to x+1)$, where δf is in the sense of the variation defined in $\mathbb{D}$.*

Proposition A.18 (Nguyen et al. [43])**.** *The following composition rules (chain rules) hold:*

1. *For $\mathbb{B} \xrightarrow{f} \mathbb{B} \xrightarrow{g} \mathbb{D}$: $(g \circ f)'(x) = \mathbf{xnor}(g'(f(x)), f'(x))$, $\forall x \in \mathbb{B}$.*
2. *For $\mathbb{B} \xrightarrow{f} \mathbb{Z} \xrightarrow{g} \mathbb{D}$, $x \in \mathbb{B}$, if $|f'(x)| \leq 1$ and $g'(f(x)) = g'(f(x)-1)$, then:*

$$(g \circ f)'(x) = \mathbf{xnor}(g'(f(x)), f'(x)).$$

Proof. The proof is as follows.

1. The case of $\mathbb{B} \xrightarrow{f} \mathbb{B} \xrightarrow{g} \mathbb{B}$ is obtained from Proposition A.15(3). For $\mathbb{B} \xrightarrow{f} \mathbb{B} \xrightarrow{g} \mathbb{N}$, by using Proposition A.16(1), the proof is similar to that of Proposition A.15(3).
2. By definition, we have
$$(g \circ f)'(x) = \mathbf{xnor}(\delta(x \to \neg x), \delta g(f(x) \to f(\neg x))). \tag{24}$$
Using property (1) of Proposition A.16, we have:
$$\begin{aligned} f(\neg x) &= f(x) + \delta f(x \to \neg x) \\ &= f(x) + \mathbf{xnor}(\delta(x \to \neg x), f'(x)). \end{aligned} \tag{25}$$
Applying Eq. 25 back to Eq. 24, the result is trivial if $f'(x) = 0$. The remaining case is $|f'(x)| = 1$ for which we have $\mathbf{xnor}(\delta(x \to \neg x), f'(x)) = \pm 1$. First, for $\mathbf{xnor}(\delta(x \to \neg x), f'(x)) = 1$, we have:
$$\begin{aligned} \delta g(f(x) \to f(\neg x)) &= \delta g(f(x) \to f(x)+1) \\ &= g'(f(x)) \\ &= \mathbf{xnor}(g'(f(x)), 1) \\ &= \mathbf{xnor}(g'(f(x)), \mathbf{xnor}(\delta(x \to \neg x), f'(x))). \end{aligned} \tag{26}$$
Substitute Eq. 26 back to Eq. 24, we obtain:
$$\begin{aligned} (g \circ f)'(x) &= \mathbf{xnor}(\delta(x \to \neg x), \delta g(f(x) \to f(\neg x))) \\ &= \mathbf{xnor}(\delta(x \to \neg x), \mathbf{xnor}(g'(f(x)), \mathbf{xnor}(\delta(x \to \neg x), f'(x)))) \\ &= \mathbf{xnor}(g'(f(x)), f'(x)), \end{aligned}$$
where that last equality is by the associativity of $\mathbf{xnor}$ and that $\mathbf{xnor}(x, x) = \text{True}$ for $x \in \mathbb{B}$. Similarly, for $\mathbf{xnor}(\delta(x \to \neg x), f'(x)) = -1$, we have:
$$\begin{aligned} \delta g(f(x) \to f(\neg x)) &= \delta g(f(x) \to f(x)-1) \\ &= -g'(f(x)-1) \\ &= \mathbf{xnor}(g'(f(x)-1), -1) \\ &= \mathbf{xnor}(g'(f(x)-1), \mathbf{xnor}(\delta(x \to \neg x), f'(x))). \end{aligned} \tag{27}$$
Substitute Eq. 27 back to Eq. 24 and use the assumption that $g'(f(x)) = g'(f(x)-1)$, we have:
$$\begin{aligned} (g \circ f)'(x) &= \mathbf{xnor}(\delta(x \to \neg x), \delta g(f(x) \to f(\neg x))) \\ &= \mathbf{xnor}(\delta(x \to \neg x), \mathbf{xnor}(g'(f(x)-1), \mathbf{xnor}(\delta(x \to \neg x), f'(x)))) \\ &= \mathbf{xnor}(g'(f(x)), f'(x)). \end{aligned}$$
Hence the preposition is proved. □

Example A.19. From Example A.14, we have $\delta\mathbf{xor}(x, a)/\delta x = \neg a$ for $a, x \in \mathbb{B}$. Using Proposition A.15-(2) we have: $\delta\mathbf{xnor}(x, a)/\delta x = a$ since $\mathbf{xnor}(x, a) = \neg\mathbf{xor}(x, a)$.

A.2.3. Extention to Multivariate Case

The properties of Boolean variation described above can be extended to the multivariate case in a straightforward manner. For example, in the case of multivariate Boolean functions, the extension is as follows.

***Definition* A.20.** *For* $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{B}^n$*, denote* $\mathbf{x}_{\neg i} \triangleq (x_1, \ldots, x_{i-1}, \neg x_i, x_{i+1}, \ldots, x_n)$ *for* $n \geq 1$ *and* $1 \leq i \leq n$*. For* $f \in \mathcal{F}(\mathbb{B}^n, \mathbb{B})$*, the (partial) variation of* f *w.r.t.* x_i*, denoted* $f'_i(\mathbf{x})$ *or* $\delta f(\mathbf{x})/\delta x_i$*, is defined as:* $f'_i(\mathbf{x}) \equiv \delta f(\mathbf{x})/\delta x_i \triangleq \mathbf{xnor}(\delta(x_i \to \neg x_i), \delta f(\mathbf{x} \to \mathbf{x}\neg i))$.

The composition rule then becomes:

Proposition A.21 (Nguyen et al. [43])**.** *Let* $f \in \mathcal{F}(\mathbb{B}^n, \mathbb{B})$*,* $n \geq 1$*, and* $g \in \mathcal{F}(\mathbb{B}, \mathbb{B})$*. For* $1 \leq i \leq n$*:*

$$(g \circ f)'_i(\mathbf{x}) = \mathbf{xnor}(g'(f(\mathbf{x})), f'_i(\mathbf{x})), \quad \forall \mathbf{x} \in \mathbb{B}^n. \tag{28}$$

Example A.22. Apply Proposition A.16-(3) to $\mathbf{Y}^{(l)}_{[k,j]}$ from Eq. 18: $\delta \mathbf{Y}^{(l)}_{[k,j]}/\delta \mathbf{W}^{(l)}_{[i,j]} = \delta \mathrm{L}(\mathbf{X}^{(l)}_{[k,i]}, \mathbf{W}^{(l)}_{[i,j]})/\delta \mathbf{W}^{(l)}_{[i,j]}$ and $\delta \mathbf{Y}^{(l)}_{[k,j]}/\delta \mathbf{X}^{(l)}_{[k,i]} = \delta \mathrm{L}(\mathbf{X}^{(l)}_{[k,i]}, \mathbf{W}^{(l)}_{[i,j]})/\delta \mathbf{X}^{(l)}_{[k,i]}$. Then, for $\mathrm{L} = \mathbf{xnor}$ as an example, we have: $\delta \mathbf{Y}^{(l)}_{[k,j]}/\delta \mathbf{W}^{(l)}_{[i,j]} = \mathbf{X}^{(l)}_{[k,i]}$ and $\delta \mathbf{Y}^{(l)}_{[k,j]}/\delta \mathbf{X}^{(l)}_{[k,i]} = \mathbf{W}^{(l)}_{[i,j]}$.

A.3. Boolean Backpropagation

This section presents how to apply the above principles of Boolean variation to define backpropagation for Boolean neural networks. The l-th layer (Eq. 18), receives the backpropagation signal from the downstream layer $l+1$. Specifically, $\mathbf{Z}^{(l)}_{[k,j]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{Y}^{(l)}_{[k,j]}}$ denotes the variation of the loss function $\mathcal{L}$ w.r.t. the output at layer l. To optimize the Boolean weights, we need to compute the corresponding loss signal, denoted as $\mathbf{Q}^{(l)}_{[i,j]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{W}^{(l)}_{[i,j]}}$. In addition, we also have to compute the loss signal for the upstream layer, defined as $\mathbf{P}^{(l)}_{[k,i]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{X}^{(l)}_{[k,i]}}$. Hereafter, we consider the logic gate $\mathrm{L} = \mathbf{xnor}$ as a concrete example.

First, using Proposition A.15, Proposition A.16, Proposition A.18 and its extension to the multivariate case by Proposition A.21 in the same manner as shown in Example A.22, we have:

$$\frac{\delta \mathbf{Y}^{(l)}_{[k,j]}}{\delta \mathbf{W}^{(l)}_{[i,j]}} = \frac{\delta \mathbf{xnor}(\mathbf{X}^{(l)}_{[k,i]}, \mathbf{W}^{(l)}_{[i,j]})}{\delta \mathbf{W}^{(l)}_{[i,j]}} = \mathbf{X}^{(l)}_{[k,i]} \tag{29}$$

$$\frac{\delta \mathbf{Y}^{(l)}_{[k,j]}}{\delta \mathbf{X}^{(l)}_{[k,i]}} = \frac{\delta \mathbf{xnor}(\mathbf{X}^{(l)}_{[k,i]}, \mathbf{W}^{(l)}_{[i,j]})}{\delta \mathbf{X}^{(l)}_{[k,i]}} = \mathbf{W}^{(l)}_{[i,j]} \tag{30}$$

Using the chain rules given by Proposition A.18, we have the following atomic variations:

$$\mathbf{Q}^{(l)}_{[k,i,j]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{W}^{(l)}_{[i,j]}}|_k = \mathbf{xnor}\left(\frac{\delta \mathcal{L}}{\delta \mathbf{Y}^{(l)}_{[k,j]}}, \frac{\delta \mathbf{Y}^{(l)}_{[k,j]}}{\delta \mathbf{W}^{(l)}_{[i,j]}}\right) = \mathbf{xnor}\left(\mathbf{Z}^{(l)}_{[k,j]}, \mathbf{X}^{(l)}_{[k,i]}\right), \tag{31}$$

$$\mathbf{P}^{(l)}_{[k,i,j]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{X}^{(l)}_{[k,i]}}|_j = \mathbf{xnor}\left(\frac{\delta \mathcal{L}}{\delta \mathbf{Y}^{(l)}_{[k,j]}}, \frac{\delta \mathbf{Y}^{(l)}_{[k,j]}}{\delta \mathbf{X}^{(l)}_{[k,i]}}\right) = \mathbf{xnor}\left(\mathbf{Z}^{(l)}_{[k,j]}, \mathbf{W}^{(l)}_{[i,j]}\right). \tag{32}$$

The variations $\mathbf{Q}^{(l)}_{[i,j]}$ and $\mathbf{G}^{(l)}_{[k,i]}$ can be then obtained by aggregating the above atomic variations over the batch dimension k and output dimension j, respectively. More specifically, denote $\mathbf{1}(\cdot)$ the indicator function. Additionally, for $b \in \mathbb{B}$ and a

variable x, we define $\mathbf{1}(x = b) = 1$ if $x_{\text{logic}} = b$ and $\mathbf{1}(x = b) = 0$ otherwise. Then, we have:

$$\mathbf{Q}^{(l)}_{[i,j]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{W}^{(l)}_{[i,j]}} = \sum_k \mathbf{1}(\mathbf{Q}^{(l)}_{[k,i,j]} = \text{TRUE})|\mathbf{Q}^{(l)}_{[k,i,j]}| - \sum_k \mathbf{1}(\mathbf{Q}^{(l)}_{[k,i,j]} = \text{FALSE})|\mathbf{Q}^{(l)}_{[k,i,j]}|, \tag{33}$$

$$\mathbf{P}^{(l)}_{[i,j]} \triangleq \frac{\delta \mathcal{L}}{\delta \mathbf{X}^{(l)}_{[k,i]}} = \sum_j \mathbf{1}(\mathbf{P}^{(l)}_{[k,i,j]} = \text{TRUE})|\mathbf{P}^{(l)}_{[k,i,j]}| - \sum_j \mathbf{1}(\mathbf{P}^{(l)}_{[k,i,j]} = \text{FALSE})|\mathbf{P}^{(l)}_{[k,i,j]}|. \tag{34}$$

A.4. Boolean Optimizer

Algorithm 1: Boolean learning process for a linear layer.

Input : Learning rate η, number of iterations T;
Initialize : $\mathbf{M}^{(l),0}_{[i,j]} = 0$; $\beta^0 = 1$;
for $t = 0, \ldots, T-1$ **do**
  /* 1. Forward */
  Compute $\mathbf{Y}^{(l),t}$ following Eq. 18;
  /* 2. Backward */
  Receive $\frac{\delta \mathcal{L}}{\delta \mathbf{Y}^{(l),t}_{[k,j]}}$ from downstream layer;
  /* 2.1 Backpropagation */
  Compute and backpropagate $\mathbf{P}^{(l),t}$ to the upstream following Eq. 34;
  /* 2.2 Weight update process */
  $N_{\text{total}} := 0$, $N_{\text{unchanged}} := 0$;
  foreach $\mathbf{W}^l_{i,j}$ **do**
    Compute $\mathbf{Q}^{(l),t+1}_{[i,j]}$ following Eq. 33;
    Update $\mathbf{M}^{(l),t+1}_{[i,j]} = \beta^t \mathbf{M}^{(l),t}_{[i,j]} + \eta^t \mathbf{Q}^{(l),t+1}_{[i,j]}$;
    $N_{\text{total}} \leftarrow N_{\text{total}} + 1$;
    if $\mathbf{xnor}(\mathbf{M}^{(l),t+1}_{[i,j]}, \mathbf{W}^{(l),t}_{[i,j]}) = \text{TRUE}$ **then**
      /* Flip weight */
      $\mathbf{W}^{(l),t+1}_{[i,j]} = \neg \mathbf{W}^{(l),t}_{[i,j]}$;
      /* Reset corresponding accumulator */
      $\mathbf{M}^{(l),t+1}_{[i,j]} = 0$;
    else
      /* Weight is unchanged */
      $\mathbf{W}^{(l),t+1}_{[i,j]} = \mathbf{W}^{(l),t}_{[i,j]}$;
      /* Update statistics to update β */
      $N_{\text{unchanged}} \leftarrow N_{\text{unchanged}} + 1$;
  Update η^{t+1}, $\beta^{t+1} = N_{\text{unchanged}}/N_{\text{total}}$;

Given the above variations, the rule for updating the Boolean weight $\mathbf{W}^{(l)}_{[i,j]}$ to minimize the loss function $\mathcal{L}$ is as follows:

$$\mathbf{W}^{(l)}_{[i,j]} = \neg \mathbf{W}^{(l)}_{[i,j]} \quad \text{if } \mathbf{xnor}\left(\mathbf{Q}^{(l)}_{[i,j]}, \mathbf{W}^{(l)}_{[i,j]}\right) = \text{TRUE}. \tag{35}$$

Based on this update rule, we can develop an optimizer that accumulates the signal $\mathbf{Q}^{(l)}_{[i,j]}$ over training iterations. Specifically, let $\mathbf{W}^{(l),t}_{[i,j]}$ denotes the weight at iteration t, and $\mathbf{M}^{(l),t}_{[i,j]}$ represents its accumulator, initialized as $\mathbf{M}^{(l),0}_{[i,j]} = 0$. The update rule for the accumulator is then defined as: The update rule for the accumulator is then defined as:

$$\mathbf{M}^{(l),t+1}_{[i,j]} \leftarrow \beta^t \mathbf{M}^{(l),t}_{[i,j]} + \eta \mathbf{Q}^{(l),t}_{[i,j]}, \tag{36}$$

where η is the accumulation factor acting as a learning rate, and β^t is an auto-regularizing factor that reflects the system's state at time t. In our work, we use brain plasticity [19] and Hebbian theory [21] to adaptively set β^t, that force the weights

to adapt to their neighborhood during. For the chose weight's neighborhood, for instance, neuron, layer, or network level, β^t is set as:

$$\beta^t = \frac{\text{Number of unchanged weights at } t}{\text{Total number of weights}}. \tag{37}$$

It to temper the importance of weight variational according to how much neurons have changed. In our experiments, β^t is set to per-layer basis and initialized as $\beta^0 = 1$ The learning process for a linear layer is described in Algorithm 1.

B. Code Samples of Core Implementation

B.1. Boolean Linear Layer and Optimizer

In this section, we provide example Python code for implementing a Boolean linear layer based on the **xor** logic gate. This implementation is based on the PyTorch framework [45]. As done in [43], the class definition for the Boolean linear layer is presented in Algorithm 2, and its backpropagation mechanism—customized via PyTorch's `autograd` system—is detailed in Algorithm 3. Each Boolean kernel is primarily implemented using this Boolean linear layer.

We consider both cases of the incoming backpropagation signal: Boolean-valued (see Algorithm 4), and real-valued (see Algorithm 5). The latter is the main use case in this paper. An example implementation of the Boolean optimizer used to update the layer's parameters is provided in Algorithm 6.

Algorithm 2: Python code of XOR linear layer

```python
import torch

from torch import Tensor, nn, autograd
from typing import Any, List, Optional, Callable


class XORLinear(nn.Linear):

    def __init__(self, in_features: int, out_features: int, bool_bprop: bool, **kwargs):
        super(XORLinear, self).__init__(in_features, out_features, **kwargs)
        self.bool_bprop = bool_bprop

    def reset_parameters(self):
        self.weight = nn.Parameter(torch.randint(0, 2, self.weight.shape))

        if self.bias is not None:
            self.bias = nn.Parameter(torch.randint(0, 2, (self.out_features,)))

    def forward(self, X):
        return XORFunction.apply(X, self.weight, self.bias, self.bool_bprop)
```

Algorithm 3: Python code of the backpropagation logic of XOR linear layer

```python
class XORFunction(autograd.Function):

    @staticmethod
    def forward(ctx, X, W, B, bool_bprop: bool):
        ctx.save_for_backward(X,W,B)
        ctx.bool_bprop = bool_bprop

        # Elementwise XOR logic
        S = torch.logical_xor(X[:,None,:], W[None,:,:])

        # Sum over the input dimension
        S = S.sum(dim=2) + B

        # 0-centered for use with BatchNorm when preferred
        S = S - W.shape[1]/2

        return S

    @staticmethod
    def backward(ctx, Z):
        if ctx.bool_bprop:
            G_X, G_W, G_B = backward_bool(ctx, Z)
        else:
            G_X, G_W, G_B = backward_real(ctx, Z)

        return G_X, G_W, G_B, None
```

Algorithm 4: Backpropagation logic with Boolean received backpropagation

```
def backward_bool(ctx, Z):
    """
    Variation of input:
        - delta(xor(x,w))/delta(x) = neg w
        - delta(Loss)/delta(x) = xnor(z,neg w) = xor(z,w)
    Variation of weights:
        - delta(xor(x,w))/delta(w) = neg x
        - delta(Loss)/delta(x) = xnor(z,neg x) = xor(z,x)
    Variation of bias:
        - bias = xnor(bias,True) ==> Variation of bias is driven in
          the same basis as that of weight with xnor logic and input True.
    Aggregation:
        - Count the number of TRUEs = sum over the Boolean data
        - Aggr = TRUEs - FALSEs = TRUEs - (TOT - TRUEs) = 2TRUES - TOT
          where TOT is the size of the aggregated dimension
    """
    X, W, B = ctx.saved_tensors

    # Boolean variation of input
    G_X = torch.logical_xor(Z[:,:,None], W[None,:,:])

    # Aggregate over the out_features dimension
    G_X = 2 * G_X.sum(dim=1) - W.shape[0]

    # Boolean variation of weights
    G_W = torch.logical_xor(Z[:,:,None], X[:,None,:])

    # Aggregate over the batch dimension
    G_W = 2 * G_W.sum(dim=0) - X.shape[0]

    # Boolean variation of bias
    if B is not None:
        # Aggregate over the batch dimension
        G_B = 2 * Z.sum(dim=0) - Z.shape[0]

    # Return
    return G_X, G_W, G_B
```

Algorithm 5: Backpropagation logic with real received backpropagation

```
def backward_real(ctx, Z):
    X, W, B = ctx.saved_tensors

    """
    Boolean variation of input processed using torch avoiding loop:
        -> xor(Z: Real, W: Boolean) = -Z * emb(W)
        -> emb(W): T->1, F->-1 => emb(W) = 2W-1
        => delta(Loss)/delta(X) = Z*(1-2W) """
    G_X = Z.mm(1-2*W)

    """
    Boolean variation of weights processed using torch avoiding loop:
        -> xor(Z: Real, X: Boolean) = -Z * emb(X)
        -> emb(X): T->1, F->-1 => emb(X) = 2X-1
        => delta(Loss)/delta(W) = Z^T * (1-2X) """
    G_W = Z.t().mm(1-2*X)

    """ Boolean variation of bias """
    if B is not None:
        G_B = Z.sum(dim=0)

    # Return
    return G_X, G_W, G_B
```

Algorithm 6: Python code of Boolean optimizer

```python
class BooleanOptimizer(torch.optim.Optimizer):

    def __init__(self, params, lr: float):
        super(BooleanOptimizer, self).__init__(params, dict(lr=lr))
        for param_group in self.param_groups:
            param_group['accums'] = [torch.zeros_like(p.data) for p in param_group['params']]
            param_group['ratios'] = [0 for p in param_group['params']]
        self._nb_flips = 0

    @property
    def nb_flips(self):
        n = self._nb_flips
        self._nb_flips = 0
        return n

    def step(self):
        for param_group in self.param_groups:
            for idx, p in enumerate(param_group['params']):
                self.update(p, param_group, idx)

    def update(self, param: Tensor, param_group: dict, idx: int):
        accum = param_group['ratios'][idx] * param_group['accums'][idx] + param_group['lr'] * param.grad.
    data
        param_group['accums'][idx] = accum
        param_to_flip = accum * (2*param.data-1) >= 1
        param.data[param_to_flip] = torch.logical_not(param.data[param_to_flip])
        param_group['accums'][idx][param_to_flip] = 0.
        param_group['ratios'][idx] = 1 - param_to_flip.float().mean()
        self._nb_flips += float(param_to_flip.float().sum())
```

B.2. Successive SVID for Kernel Extraction

Algorithm 7 illustrate the Python code of the SVID algortithm to extract the optimal Boolean weights and scaling factors for one kernel. Based on this, Algorithm 8 illustrates the succesive SVID algorithm to extract all kernels.

Algorithm 7: Python code of SVID approximation of a FP matrix.

```python
def svid_approximation(w):
    """
    Approximate the input matrix `w` by a boolean matrix and a rank-1 matrix:
        w = w_bool * (s_out * s_in.T)

    Args:
        w (torch.Tensor): Input tensor of shape (*, m, n).

    Returns:
        tuple:
            - w_bool (torch.Tensor): Boolean matrix of the same shape as `w`.
            - w_res (torch.Tensor): Residual matrix, w - w_bool * (s_out * s_in.T).
            - s_in (torch.Tensor): Scaled first left singular vector of `w`.
            - s_out (torch.Tensor): Scaled first right singular vector of `w`.
    """
    U, S, Vh = torch.linalg.svd(abs(w.data.clone().float()), full_matrices=False)

    w_bool = torch.sign(w)
    s_in = torch.sqrt(S[0]) * Vh[0,:].reshape(1,-1)
    s_out = torch.sqrt(S[0]) * U[:,0].reshape(-1,1)

    w_res = w - w_bool * torch.matmul(s_out, s_in)

    return w_bool, w_res, s_in, s_out
```

Algorithm 8: Python code of successively extracts kernels from FP matrix using SVID.

```python
def successive_svid(w_fp, n_kernels):
    """
    Perform successive SVID on the input matrix to extract Boolean kernels.

    Args:
        w_fp (torch.Tensor): Input weight matrix.
        n_kernels (int): Number of iterations to extract kernels.

    Returns:
        list: List of dictionaries containing `n_kernels` kernels, each has:
            - w_bool (torch.Tensor): Boolean matrix.
            - s_in (torch.Tensor): Input scaling vector.
            - s_out (torch.Tensor): Output scaling vector.
    """
    boolean_kernels = []

    w = w_fp # The input to SVID at first iteration is the original weight

    for k in range(n_kernels):
        # Extract the Boolean weights, residual, and scaling vectors
        w_bool, w_res, s_in, s_out = svid_approximation(w)

        # Save the extracted kernel
        boolean_kernels.append({'w_bool': w_bool, 's_in': s_in, 's_out': s_out})

        # The input to SVID for the next iteration is the current residual matrix
        w = w_res

    return boolean_kernels
```

C. Proof of Propositions

For completeness, we include the proofs of Propositions related to SVID approximation used in the main paper.

C.1. Proof of Boolean Linear Reformulation using SVID

Proposition C.1 (Xu et al. [58]). *Given the weight matrix $\mathbf{W}_{\mathrm{FP}}$ and input $\mathbf{X}$, the linear layer can be reformulated as the following using* SVID *approximation, $\mathbf{W}_{\mathrm{FP}} \approx \mathbf{W}_{\mathrm{bool}} \odot \left(\mathbf{s}_{\mathrm{out}}\mathbf{s}_{\mathrm{in}}^{\top}\right)$, as follows:*

$$\mathbf{X}\mathbf{W}_{\mathrm{FP}}^{\top} \approx \left[\left(\mathbf{X} \odot \mathbf{s}_{\mathrm{in}}^{\top}\right)\mathbf{W}_{\mathrm{bool}}^{\top}\right] \odot \mathbf{s}_{\mathrm{out}}^{\top}. \tag{38}$$

Proof. Due to the SVID approximation, we have $\mathbf{W}_{\mathrm{FP}[i,j]} \approx \mathbf{W}_{\mathrm{bool}[i,j]}\mathbf{s}_{\mathrm{out}[i]}\mathbf{s}_{\mathrm{in}[j]}$. Then, we have:

$$\begin{aligned}
\left(\mathbf{X}\mathbf{W}_{\mathrm{FP}}^{\top}\right)_{[i,j]} &\approx \sum_k \mathbf{X}_{[i,k]}\mathbf{W}_{\mathrm{FP}[k,j]}^{\top} && (39)\\
&= \sum_k \mathbf{X}_{[i,k]}\mathbf{W}_{\mathrm{FP}[j,k]} && (40)\\
&= \sum_k \mathbf{X}_{[i,k]}\mathbf{W}_{\mathrm{bool}[j,k]}\mathbf{s}_{\mathrm{out}[j]}\mathbf{s}_{\mathrm{in}[k]} && (41)\\
&= \sum_k \mathbf{X}_{[i,k]}\mathbf{s}_{\mathrm{in}[k]}\mathbf{W}_{\mathrm{bool}[j,k]}\mathbf{s}_{\mathrm{out}[j]} && (42)\\
&= \sum_k \left(\mathbf{X} \odot \mathbf{s}_{\mathrm{in}}^{\top}\right)_{[i,k]}\mathbf{W}_{\mathrm{bool}[k,j]}^{\top}\mathbf{s}_{\mathrm{out}[j]} && (43)\\
&= \left[\left(\mathbf{X} \odot \mathbf{s}_{\mathrm{in}}^{\top}\right)\mathbf{W}_{\mathrm{bool}}^{\top}\right]_{[i,j]}\mathbf{s}_{\mathrm{out}[j]} && (44)\\
&= \left\{\left[\left(\mathbf{X} \odot \mathbf{s}_{\mathrm{in}}^{\top}\right)\mathbf{W}_{\mathrm{bool}}^{\top}\right] \odot \mathbf{s}_{\mathrm{out}}^{\top}\right\}_{[i,j]}. && (45)
\end{aligned}$$

Thus, the proposition is proved. □

C.2. Proof of Proposition 4.1

Lemma C.2 (Xu et al. [58]). *Denote $\sigma_i(\mathbf{W})$ the i-th biggest singular value of matrix $\mathbf{W}$. The following inequality holds:*

$$\sigma_1(|\mathbf{W}|) \geq \sigma_1(\mathbf{W}). \tag{46}$$

Proof. By the definition of induced norm, we have:

$$\sigma_1(\mathbf{W}) = \|\mathbf{W}\|_2 = \max_{\mathbf{x}, \|\mathbf{x}\|_2=1} \|\mathbf{W}\mathbf{x}\|_2, \tag{47}$$

$$\sigma_1(|\mathbf{W}|) = \||\mathbf{W}|\|_2 = \max_{\mathbf{y}, \|\mathbf{y}\|_2=1} \||\mathbf{W}|\mathbf{y}\|_2. \tag{48}$$

In addition, because $\forall \mathbf{x}, \|\mathbf{x}\|_2 = 1$, we have:

$$\begin{aligned}
\||\mathbf{W}||\mathbf{x}|\|_2^2 &= \sum_i \left(\sum_j |\mathbf{W}_{[i,j]}||\mathbf{x}_{[j]}| \right)^2 && (49)\\
&\geq \sum_i \left(|\sum_j \mathbf{W}_{[i,j]}\mathbf{x}_{[j]}| \right)^2 && (50)\\
&= \sum_i \left(\sum_j \mathbf{W}_{[i,j]}\mathbf{x}_{[j]} \right)^2 && (51)\\
&= \|\mathbf{W}\mathbf{x}\|_2^2. && (52)
\end{aligned}$$

Therefore

$$\begin{aligned}
\max_{\mathbf{y}, \|\mathbf{y}\|_2=1} \||\mathbf{W}|\mathbf{y}\|_2 &\geq \max_{\mathbf{x}, \|\mathbf{x}\|_2=1} \|\mathbf{W}\mathbf{x}\|_2 && (53)\\
\Leftrightarrow \sigma_1(|\mathbf{W}|) &\geq \sigma_1(\mathbf{W}). && (54)
\end{aligned}$$

Thus, the lemma is proved. □

Proposition C.3 (Restated from Xu et al. [58]). *For $\mathbf{W} \in \mathbb{R}^{m\times n}$, write $\mathbf{W} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\widetilde{\mathbf{V}}^\top$ its* SVD. *Let $\mathbf{a} = \sqrt{\tilde{\sigma}_1}\widetilde{\mathbf{U}}_{[:,1]}$, and $\mathbf{b} = \sqrt{\tilde{\sigma}_1}\widetilde{\mathbf{V}}_{[:,1]}$. Similarly, denote $|\mathbf{W}| = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ its* SVD*; $\mathbf{s}_{\text{in}}$ and $\mathbf{s}_{\text{out}}$ are given as: $\mathbf{s}_{\text{in}} = \sqrt{\sigma_1}\mathbf{V}_{[:,1]}$, and $\mathbf{s}_{\text{out}} = \sqrt{\sigma_1}\mathbf{U}_{[:,1]}$. We decompose the matrix as $\mathbf{W} = \mathbf{W}_{\text{bool}} \odot |\mathbf{W}| \approx \mathbf{W}_{\text{bool}} \odot \left(\mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right)$. We then have:*

$$\left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right\|_F^2 \leq \left\|\mathbf{W} - \mathbf{a}\mathbf{b}^\top\right\|_F^2. \tag{55}$$

Proof. We denote the following error matrices:

$$\begin{aligned}
\mathbf{E}_1 &= \mathbf{W} - \mathbf{a}\mathbf{b}^\top, && (56)\\
\mathbf{E}_2 &= |\mathbf{W}| - \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top. && (57)
\end{aligned}$$

Multiplying $\mathbf{W}_{\text{bool}}$ with both sides of Eq. 57, we have:

$$\begin{aligned}
\mathbf{W}_{\text{bool}} \odot |\mathbf{W}| - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top &= \mathbf{W}_{\text{bool}} \odot \mathbf{E}_2 && (58)\\
\Leftrightarrow \mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top &= \mathbf{W}_{\text{bool}} \odot \mathbf{E}_2. && (59)
\end{aligned}$$

Thus, we have:

$$\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\|_F^2 = \|\mathbf{W}_{\text{bool}} \odot \mathbf{E}_2\|_F^2 \tag{60}$$

$$= \sum_{i,j} \mathbf{W}_{\text{bool}[i,j]}^2 + \mathbf{E}_{2[i,j]}^2 \tag{61}$$

$$= \sum_{i,j} \mathbf{E}_{2[i,j]}^2 \tag{62}$$

$$= \|\mathbf{E}_2\|_F^2 \tag{63}$$

For SVD decomposition, the norm of the above error matrices in the rank-1 approximation is the um of squares of all singular values except the largest one. In particular, we have:

$$\|\mathbf{E}_1\|_F^2 = \sum_{i=2}^{n} \sigma_i^2(\mathbf{W}), \tag{64}$$

$$\|\mathbf{E}_2\|_F^2 = \sum_{i=2}^{n} \sigma_i^2(|\mathbf{W}|). \tag{65}$$

Since $\|\mathbf{W}\|_F^2 = \||\mathbf{W}|\|_F^2$, we have:

$$\sum_{i=1}^{n} \sigma_i^2(\mathbf{W}) = \sum_{i=1}^{n} \sigma_i^2(|\mathbf{W}|) \tag{66}$$

$$\Leftrightarrow \|\mathbf{E}_1\|_F^2 + \sigma_1^2(\mathbf{W}) = \|\mathbf{E}_2\|_F^2 \sigma_1^2(|\mathbf{W}|). \tag{67}$$

Thus, according to Lemma C.2 and Eq. 63, we have:

$$\|\mathbf{E}_2\|_F^2 \leq \|\mathbf{E}_1\|_F^2 \tag{68}$$

$$\left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right\|_F^2 \leq \left\|\mathbf{W} - \mathbf{a}\mathbf{b}^\top\right\|_F^2. \tag{69}$$

Thus, the proposition is proved. □

C.3. Proof of Proposition 4.3

Proposition C.4. *For $\mathbf{W} \in \mathbb{R}^{m\times n}$, we denote $|\mathbf{W}| = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ its SVD. $\mathbf{s}_{\text{in}}$ and $\mathbf{s}_{\text{out}}$ are given as: $\mathbf{s}_{\text{in}} = \sqrt{\sigma_1}\mathbf{V}_{[:,1]}$, and $\mathbf{s}_{\text{out}} = \sqrt{\sigma_1}\mathbf{U}_{[:,1]}$. We decompose the matrix as $\mathbf{W} = \mathbf{W}_{\text{bool}} \odot |\mathbf{W}| \approx \mathbf{W}_{\text{bool}} \odot \left(\mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right)$. We then have:*

$$\left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right\|_F^2 \leq \left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{c}\mathbf{d}^\top\right\|_F^2, \quad \forall \mathbf{c} \in \mathbb{R}^{m\times 1}, \forall \mathbf{d} \in \mathbb{R}^{n\times 1}. \tag{70}$$

Proof. Similar to the proof of Proposition 4.3, we denote the following error matrices $\mathbf{E}_1 = |\mathbf{W}| - \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top$ and $\mathbf{E}_2 = |\mathbf{W}| - \mathbf{c}\mathbf{d}^\top$. We have that

$$\mathbf{W}_{\text{bool}} \odot |\mathbf{W}| - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top = \mathbf{W}_{\text{bool}} \odot \mathbf{E}_1 \tag{71}$$

$$\Leftrightarrow \mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top = \mathbf{W}_{\text{bool}} \odot \mathbf{E}_1. \tag{72}$$

Therefore,

$$\left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right\|_F^2 = \|\mathbf{W}_{\text{bool}} \odot \mathbf{E}_1\|_F^2 = \sum_{i,j} \mathbf{W}_{\text{bool}[i,j]}^2 \mathbf{E}_{1[i,j]}^2 = \sum_{i,j} \mathbf{E}_{1[i,j]}^2 = \|\mathbf{E}_1\|_F^2. \tag{73}$$

Similarly, we have that

$$\left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{a}\mathbf{b}^\top\right\|_F^2 = \|\mathbf{E}_2\|_F^2. \tag{74}$$

Thus, we need to show that

$$\|\mathbf{E}_1\|_F^2 \leq \|\mathbf{E}_2\|_F^2 \tag{75}$$

Additionally, we denote the rank-k approximation to $|\mathbf{W}|$ by SVD as $\mathbf{S}_k$:

$$\mathbf{S}_k = \sum_{i=1}^{k} \sigma_i \mathbf{U}_{[:,i]} \mathbf{V}_{[:,i]}^\top. \tag{76}$$

With this notation, we have that $\mathbf{S}_1 = \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top$ is the rank-1 approximation of $|\mathbf{W}|$ by SVD.

From Eq. 75, we need to show that if there is an arbitrary rank-1 approximation to $|\mathbf{W}|$, $\mathbf{P}_1 = \mathbf{c}\mathbf{d}^\top$, we then have

$$\left\||\mathbf{W}| - \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right\|_F^2 \leq \left\||\mathbf{W}| - \mathbf{c}\mathbf{d}^\top\right\|_F^2. \tag{77}$$

This can be done by using the Eckart-Young-Mirsky theorem [17]. First, we have that

$$\||\mathbf{W}| - \mathbf{S}_1\|_F^2 = \left\||\mathbf{W}| - \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right\|_F^2 = \left\|\sum_{i=2}^{n} \sigma_i \mathbf{U}_{[:,i]} \mathbf{V}_{[:,i]}^\top\right\|_F^2 = \sum_{i=2}^{n} \sigma_i^2. \tag{78}$$

By the triangle inequality with the spectral norm, if $|\mathbf{W}| = \mathbf{C} + \mathbf{D}$ then $\sigma_1(|\mathbf{W}|) \leq \sigma_1(\mathbf{C}) + \sigma_1(\mathbf{D})$. Suppose the $\mathbf{C}_k$ and $\mathbf{D}_k$ denote the rank-k approximation to $\mathbf{C}$ and $\mathbf{D}$ by SVD method, respectively. Then, for any $i, j \geq 1$ we have

$$\begin{aligned}
\sigma_i(\mathbf{C}) + \sigma_j(\mathbf{D}) &= \sigma_1(\mathbf{C} - \mathbf{C}_{i-1}) + \sigma_1(\mathbf{D} - \mathbf{D}_{j-1}) && (79)\\
&\geq \sigma_1(|\mathbf{W}| - \mathbf{C}_{i-1} - \mathbf{D}_{j-1}) && (80)\\
&\geq \sigma_1(|\mathbf{W}| - \mathbf{S}_{i+j-2}) \qquad (\text{since rank}(\mathbf{C}_{i-1} + \mathbf{D}_{j-1}) \leq i + j - 2) && (81)\\
&= \sigma_{i+j-1}(|\mathbf{W}|). && (82)
\end{aligned}$$

Because $\sigma_2(\mathbf{P}_1) = 0$, when $\mathbf{C} = |\mathbf{W}| - \mathbf{P}_1$ and $\mathbf{D} = \mathbf{P}_1$ we have that for $i \geq 1, j = 2$, $\sigma_i(|\mathbf{W}| - \mathbf{P}_1) \geq \sigma_{i+1}(|\mathbf{W}|)$. As a result,

$$\begin{aligned}
\||\mathbf{W}| - \mathbf{P}_1\|_F^2 = \sum i = 1^n \sigma_i(|\mathbf{W}| - \mathbf{P}_1)^2 \geq \sum i = 2^n \sigma_i(|\mathbf{W}|)^2 = \||\mathbf{W}| - \mathbf{S}_1\|_F^2 && (83)\\
\Leftrightarrow \|\mathbf{E}_2\|_F^2 \geq \|\mathbf{E}_1\|_F^2 && (84)\\
\Leftrightarrow \left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{c}\mathbf{d}^\top\right\|_F^2 \geq \left\|\mathbf{W} - \mathbf{W}_{\text{bool}} \odot \mathbf{s}_{\text{out}}\mathbf{s}_{\text{in}}^\top\right\|_F^2. && (85)
\end{aligned}$$

Hence the proposition is proved. □

D. Details on Kernel Allocation

D.1. Weight Importance Estimation

We assess the importance of a linear weight in the original FP model by comparing the representations at its input and output. Let $\mathbf{X} \in \mathbb{R}^{d \times n}$ and $\mathbf{Y} \in \mathbb{R}^{d \times m}$ denote the input and output matrices of a linear layer, respectively, where d is the number of samples, and n and m are the input and output feature dimensions. We hypothesize that a weight is important if it significantly transforms the input representations. For example, a weight matrix equivalent to the identity does not alter the representations and thus would be considered unimportant. To quantify this transformation, we use a robust metric for comparing neural representations.

Various similarity measures can be used for this purpose, such as cosine similarity, as done in [20]. In this work, we adopt PWCCA [42], which is particularly well-suited for our setting: it is invariant to linear transformations—an essential property given that large language models (LLMs) are primarily composed of linear layers—and effectively captures shared structure while filtering out noise [42].

Specifically, we define the importance score as:

$$h = 1 - \frac{1}{c}\sum_{i=1}^{c} \rho_{\mathrm{PWCCA},i}(\mathbf{X}, \mathbf{Y}), \tag{86}$$

where c denotes the number of canonical vectors used in the comparison (typically, $c = \min(n, m)$). The matrices $\mathbf{X}$ and $\mathbf{Y}$ are obtained by simply forwarding a set of data samples through the network. In our experiments, we use 128 random samples from the WikiText2 training set to estimate the importance score. Here, $\rho_{\mathrm{PWCCA},i}$ represents the projection-weighted correlation along the i-th canonical direction. The following section describes in detail how this correlation is computed.

Projection-weighted Canonical Correlation Analysis. Canonical Correlation Analysis (CCA) finds bases for two matrices such that, when the original matrices are projected onto these bases, the resulting projections are maximally correlated. Without loss of generality, we assume that $n \leq m$. For $1 \leq i \leq n$, the i-th canonical correlation coefficient ρ_i is given by:

$$\begin{aligned} \rho_i = \max_{\mathbf{w}_{\mathbf{X}}^i, \mathbf{w}_{\mathbf{Y}}^i} \mathrm{corr}(\mathbf{X}\mathbf{w}_{\mathbf{X}}^i, \mathbf{Y}\mathbf{Y}\mathbf{w}_{\mathbf{Y}}^i) \\ \text{subject to } \mathbf{X}\mathbf{w}_{\mathbf{X}}^i \perp \mathbf{X}\mathbf{w}_{\mathbf{X}}^j \quad \forall j < i \\ \mathbf{Y}\mathbf{w}_{\mathbf{Y}}^i \perp \mathbf{Y}\mathbf{w}_{\mathbf{Y}}^j \quad \forall j < i. \end{aligned} \tag{87}$$

The vectors $\mathbf{w}_{\mathbf{X}}^i \in \mathbb{R}^n$ and $\mathbf{w}_{\mathbf{Y}}^i \in \mathbb{R}^m$ that maximize ρ_i are called the canonical weights. These weights transform the original data into the canonical variables $\mathbf{X}\mathbf{w}_{\mathbf{X}}^i$ and $\mathbf{Y}\mathbf{w}_{\mathbf{Y}}^i$. The constraints in Eq. 87 enforce orthogonality among the canonical variables, ensuring that each successive pair captures a distinct mode of correlation.

The mean CCA correlation is then computed as:

$$\bar{\rho}_{\mathrm{CCA}} = \frac{\sum_{i=1}^{n} \rho_i}{n}, \tag{88}$$

where n is the number of canonical correlation coefficients considered.

CCA is sensitive to perturbation when the condition number of $\mathbf{X}$ and $\mathbf{Y}$ is large. To imporve robustness, [42] propose a strategy to reduce this sensitivity, which they term "projection-weighted CCA" (PWCCA).

$$\rho_{\mathrm{PWCCA},i} = \frac{\sum_{i=1}^{c} \alpha_i \rho_i}{\sum_{i=1}^{c} \alpha_i}, \quad \alpha_i = \sum_j |\langle \mathbf{h}_i, \mathbf{x}_j \rangle|, \tag{89}$$

where $\mathbf{x}_j$ is the j-th column of $\mathbf{X}$, and $\mathbf{h}_i = \mathbf{X}\mathbf{w}_{\mathbf{X}}^i$ is the vector of canonical variables formed by projecting $\mathbf{X}$ to the i-th canonical cooridate frame.

D.2. Kernel Allocation Algorithm

Algorithm 9 illustrates the details of our algorithm for kernel allocation.

Algorithm 9: Kernel allocation.

Input
- $T \geq 1$; /* model expansion limit */
- $\mathbf{E} = [e_l^{[k]}] \in \mathbb{R}^{N_\mathbf{W} \times K_{\max}}$ for $k \in [1, K_{\max}], l \in [1, N_\mathbf{W}]$; /* residual approx error */
- $\mathbf{h} = [h_l] \in \mathbb{R}^{N_\mathbf{W} \times 1}$; /* weight importance scores */
- $\mathbf{p} = [p_l] \in \mathbb{R}^{N_\mathbf{W} \times 1}$; /* weight size ratios */

Initialize
- $\mathbf{k} = [1, \ldots, 1]^\mathrm{T}$ of length $N_\mathbf{W}$; /* starting choice */
- $\mathbf{f} = \mathbf{k} < K_{\max}$; /* feasible indicator */
- $\mathbf{C} = \left(\frac{1}{\mathbf{p}} \log \frac{1}{\mathbf{p}}\right) \odot \mathbf{h} \odot \mathbf{E}$; /* where $\odot$ is broadcasted over $\mathbf{E}$ columns */

While *not all* $\mathbf{f}$ *is* False **do**
- $\mathbf{g} := \emptyset, \mathbf{l} := \emptyset$;
- **for** $l = 1 : N_\mathbf{W}$ **do**
  - **if** $\mathbf{f}[l] =$ True **then**
    - $g := \mathbf{C}[l, \mathbf{k}[l]] - \mathbf{C}[l, \mathbf{k}[l] + 1]$; /* gain by increasing kernel size by 1 */
    - Append l to $\mathbf{l}$, append g to $\mathbf{g}$;
- Sort $\mathbf{g}$ in decreasing order, and arrange $\mathbf{l}$ accordingly;
- **for** (g, l) *in* $(\mathbf{g}, \mathbf{l})$ **do**
  - $\mathbf{k}_l := \mathbf{k}$;
  - $\mathbf{k}_l[l] = \mathbf{k}_l[l] + 1$;
  - **if** $\mathbf{k}_l^\mathrm{T} \mathbf{p} \leq T$ **then**
    - $\mathbf{k}[l] = \mathbf{k}[l] + 1$;
    - **break** ; /* escape the for loop */
  - **else**
    - $\mathbf{f}[l] :=$ False;
- $\mathbf{f} \leftarrow \mathbf{and}(\mathbf{f}, \mathbf{k} < K_{\max})$; /* element-wise logical **and** */

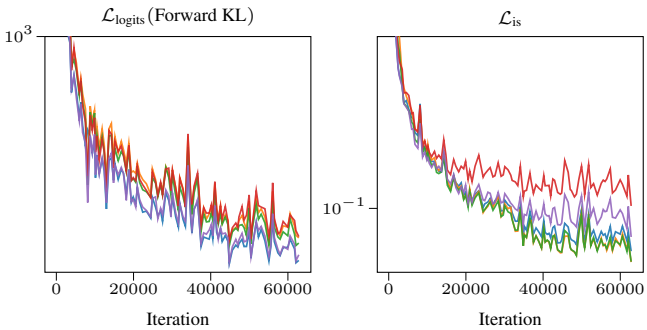
return $\mathbf{k}$

E. Additional Experiemental Results

E.1. Additional Information of Experiemental Settings

We use 12 Nvidia GPUs of Tesla V100 for our experiments. We follow exactly the experimental settings in [25]. The results of the baselines in Table 2 are taken from [58, 25].

E.2. On the Choice of KD Loss



$\mathrm{D}_{\text{logits}}$	Wiki2	C4
Forward KL	**31.39**	**28.50**
Reverse KL	33.14	29.46
Symmetric KL	32.67	29.26
JS Divergence	31.78	28.69
TV Distance	33.02	29.56

Figure 9: The training convergence of $\mathcal{L}_{\text{is}}$, and $\mathcal{L}_{\text{logits}}$, measured by Forward KL, and the final results with respect to the choice of $\mathrm{D}_{\text{logits}}$.

Fig. 9 illustrates the convergence and results of using different choices for $\mathrm{D}_{\text{logits}}$ in Eq. 13. Despite its simplicity, forward KL achieves the best performance. More complex measures, such as total variance (TV) distance [57] and Jensen-Shannon (JS) divergence [1], offer no significant benefits in our case. Furthermore, we observe that the final perplexity is strongly correlated with $\mathcal{L}_{\text{logits}}$ using forward KL, but not with $\mathcal{L}_{\text{is}}$, as shown in Fig. 9 and Fig. 6. As a result, we employ the forward KL in all experiments.

E.3. Results of Different Number of Kernels on LLMs

To complement the Table 2, Table 5 shows the benchmarking results of LLMs using our MBOK method with varying numbers of kernels per weight. Consistent with the observations made on smaller models in § 6.1.1, we observe that increasing the number of kernels generally improves performance. However, the performance gains begin to diminish noticeably beyond three kernels.

Table 5: Perplexity and zero-shot accuracy results of our MBOK method with different number of kernels.

Model	Method	Wbits	Perplexity (↓)		Zero-shot Accuracy (↑)						
			Wiki2	C4	BoolQ	PIQA	Hella.	WinoG.	ARC-e	ARC-c	Average
OPT-1.3B [63]	MBOK (2 kernels)	2×1	16.13	16.61	58.53	70.67	48.11	56.75	48.19	27.90	51.69
	MBOK (3 kernels)	3×1	15.30	15.68	60.64	70.78	50.71	56.83	48.82	28.49	52.71
	MBOK (4 kernels)	4×1	**14.83**	**14.92**	60.95	70.85	51.02	56.85	49.13	29.24	**53.01**
LLaMA-7B [53]	MBOK (2 kernels)	2×1	6.83	8.53	69.20	74.32	64.80	60.30	49.05	34.90	58.76
	MBOK (3 kernels)	3×1	6.20	7.76	67.89	76.15	68.91	63.30	48.94	37.62	60.47
	MBOK (4 kernels)	4×1	**6.01**	**7.53**	68.16	76.71	69.85	62.09	49.24	38.14	**60.70**
LLaMA-13B [53]	MBOK (2 kernels)	2×1	6.17	7.88	68.10	76.33	69.88	64.17	52.34	37.88	61.45
	MBOK (3 kernels)	3×1	5.58	7.15	67.39	77.74	73.37	66.61	54.04	41.21	63.39
	MBOK (4 kernels)	4×1	**5.38**	**6.91**	68.69	77.63	74.23	66.53	56.14	41.38	**64.10**

E.4. Additional Results on LLaMA-2

Table 6 shows the results on LLaMA2-13B [54]. Similar to the Table 2, the results of the baselines are taken from [58] and [25]. It is clear that our method consistently outperforms the baselines across different metrics and model sizes.This further emphasizes the robustness of our approach across various types of models.

Table 6: Perplexity and zero-shot accuracy results of Float16, quantized and binarized LLaMA2 models.

Model	Method	Wbits	Perplexity (↓)		Zero-shot Accuracy (↑)						
			Wiki2	C4	BoolQ	PIQA	Hella.	WinoG.	ARC-e	ARC-c	Average
LLaMA2-7B [54]	FP-16	16	5.47	6.97	71.10	76.88	72.94	67.09	53.58	40.61	63.70
	PB-LLM [61]	1.7	76.75	85.92	62.17	52.82	26.87	50.11	26.89	24.31	40.53
	BiLLM [23]	1.11	27.72	36.34	62.14	59.19	35.18	53.11	34.22	26.54	45.06
	OneBit [58]	1	8.60	10.74	63.06	70.40	54.24	56.67	40.82	29.35	52.42
	MoS [25]	1	7.88	9.75	65.02	71.55	59.41	56.18	41.84	30.03	54.01
	QPTQ [18]	2	7.7e3	NaN	42.97	49.46	26.19	50.28	26.77	28.58	37.38
	LLM-QAT [38]	2	1.1e3	6.6e2	59.14	50.12	25.10	49.08	26.26	26.96	35.89
	OmniQuant [50]	2	31.21	64.34	58.69	56.53	33.87	51.22	33.63	24.32	43.12
	MBOK [Ours]	2×1	**6.87**	**8.74**	66.94	74.97	65.59	61.72	44.82	34.21	**58.04**
	MBOK [Ours]	3×1	**6.12**	**7.81**	65.46	75.79	69.59	62.04	49.11	37.80	**59.97**
LLaMA2-13B [54]	FP-16	16	4.88	6.47	68.99	79.05	76.62	69.77	57.95	44.20	66.10
	PB-LLM [61]	1.7	155.25	151.15	37.82	53.26	28.89	49.48	28.28	23.72	36.91
	BiLLM [23]	1.11	20.71	27.19	62.20	62.51	38.05	56.35	40.69	27.73	47.92
	OneBit [58]	1	7.56	9.67	65.66	71.60	60.07	56.91	45.76	31.74	55.29
	MoS [25]	1	7.08	8.91	66.12	73.72	63.80	58.98	45.71	33.19	57.09
	QPTQ [18]	2	2.1e3	3.2e2	40.61	51.74	25.67	51.85	25.46	27.30	37.11
	LLM-QAT [38]	2	5.1e2	1.1e3	39.85	49.08	24.37	51.38	27.15	24.32	36.03
	OmniQuant [50]	2	16.88	27.02	62.05	62.24	50.34	53.20	40.66	29.61	49.68
	MBOK [Ours]	2×1	**5.97**	**7.85**	66.32	75.84	70.24	62.51	50.00	37.46	**60.40**
	MBOK [Ours]	2×1	**5.35**	**7.07**	66.80	77.59	73.79	65.27	54.47	41.89	**63.30**

E.5. Generation Quality

Table 7 illustrates the generated text given zero-shot arbitrary questions from LLaMA-13B with FP-16 weights and MBOK using three kernels. It is clearly that our model is able to produce meaningful and coherennt text compared to the original FP-16 model.

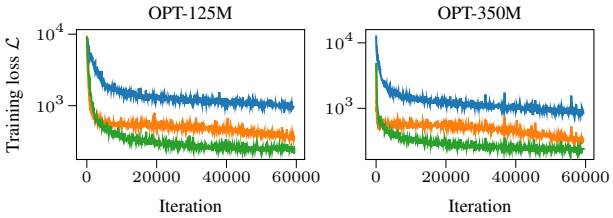
Table 7: Comparison of generation quality (*non-curated*) on LLaMA-13B models with FP-16 and MBOK using three kernels.

Prompt:	What are the capitals of China, US and Russia?
FP16:	The capital of China is Beijing. The capital of the United States is Washington D.C. and the capital of Russia is Moscow.
MKOB:	The capital of China is Beijing. The capital of the US is Washington D.C. and the capital of Russia is Moscow.
Prompt:	I have a friend who lives in London and I would like to visit him. How would I go there from Paris?
FP16:	You can take the Eurostar from Gare du Nord in Paris to St. Pancras Station in London. The trip takes about 2 hours and 15 minutes.
MKOB:	The fastest way to get from Paris to London is to take the Eurostar train. The Eurostar departs from the Gare du Nord station in Paris and arrives at the St. Pancras International station in London. The trip takes about 2 hours and 15 minutes.
Prompt:	What is Fete du Travail?
FP16:	It is an annual holiday celebrated all over the world that resulted from the labour union movement, to celebrate the economic and social achievements of workers. The majority of countries celebrate Labour Day on May Day, or May 1, and it is popularly known as May Day and International Workers' Day
MKOB:	In France, May 1st is a public holiday. It is also known as Labour Day, International Workers_ Day, or May Day. It is a celebration of the social and economic achievements of the labour movement. It originated from the eight-hour day movement, which advocated eight hours for work, eight for recreation, and eight for rest.
Prompt:	What are the main ingredients in pho, Vietnam_s iconic noodle soup?
FP16:	Rice noodles, beef or chicken, onions, ginger, cilantro, bean sprouts, basil, lime, hoisin sauce, Sriracha, and fish sauce.
MKOB:	Bone broth, rice noodles, fish sauce, herbs and spices, and beef, chicken, or seafood, depending on the type of pho you_re eating. Pho is usually served with bean sprouts, Thai basil, lime wedges, and hoisin and sriracha sauces.

E.6. Comparison with Latent-weight Approaches

We compare our method with latent-weight approaches on OPT models, using MoS with 3 experts and our method with 3 Boolean kernels. We also introduce a baseline using our SVID framework to construct 3 binary weights that rely on FP latent weights for training. Results in Fig. 10 show that our method converges much faster, as it directly optimizes Boolean parameters without the need for STE to approximate gradient signals. Both our approach and the latent-weight method outperform MoS, demonstrating the benefit of using additional Boolean kernels and our successive SVID framework. Our

method is also more efficient, avoiding the need for FP latent weights and extra momentum.



OPT	Method	Wiki2	C4
125M	MoS (3 experts) [25]	38.62	34.72
	3 Latent weights	29.47	27.18
	MBOK (3 kernels) [Ours]	**29.10**	**26.48**
350M	MoS (3 experts) [25]	29.93	28.25
	3 Latent weights	23.58	22.65
	MBOK (3 kernels) [Ours]	**23.12**	**22.10**

Figure 10: Comparions between our method and latent-weight approaches.