PAHQ: Accelerating Automated Circuit Discovery through Mixed-Precision Inference Optimization

Anonymous ACL submission

Abstract

Circuit discovery, which involves identifying sparse and task-relevant subnetworks in pre-trained language models, is a cornerstone of mechanistic interpretability. Automated Circuit Discovery (ACDC) has emerged as a pivotal methodology in circuit discovery, but its application to large language models is severely limited by computational inefficiency and prohibitively high memory requirements. Although several accelerated approaches have been proposed, they primarily rely on linear approximations to ACDC, which significantly compromises analytical faithfulness. Our proposed method for accelerating automated circuit discovery, Per Attention Head Quantization (PAHQ), takes a fundamentally different approach by optimizing the efficiency of each individual patching operation. PAHQ leverages a fundamental alignment between activation patching and mixed-precision quantization (MPQ): interpretability analysis through patching essentially performs targeted ablation studies. Therefore, we can maintain high precision exclusively for investigated components while safely reducing precision elsewhere in the network. PAHQ-accelerated ACDC reduces runtime by up to 80% and memory consumption by up to 30% compared to unaccelerated ACDC while maintaining faithfulness. Importantly, our method readily integrates with existing edge-based circuit discovery techniques by modifying the attention computation mechanism. This training-free approach provides a practical and novel pathway for accelerating mechanistic interpretability methods.

1 Introduction

As LLMs scale up (Minaee et al., 2025), understanding their internal mechanisms becomes crucial for effective utilization and supervision (Chen et al., 2021; Li et al., 2022; Manzari et al., 2023). A main method for understanding model behavior is to decompose a model into circuits—interpretable computational subgraphs responsible for specific model behaviors (Wang et al., 2022; Bereska and Gavves, 2024). The Automatic Circuit Discovery (ACDC) method proposed by (Conmy et al., 2023) served as the first scalable, end-to-end approach for identifying behavior-related circuits in models. While ACDC is effective, it relies on activation patching (replacing activations from one forward pass with those from another) using an expensive greedy search that ablates each edge, which limits its scalability (Bhaskar et al., 2024). One alternative is Edge Attribution Patching (EAP) (Syed et al., 2023), which replaces activation patching with gradient-based linear approximations. While EAP is significantly faster, these first-order approximations often compromise faithfulness to the full model.

Given these computational and memory constraints, along with the need to maintain model performance, quantization (Han et al., 2016; Jacob et al., 2017; Krishnamoorthi, 2018) naturally emerges as a promising approach to address these challenges. Quantization converts the high-precision floating-point parameters of the model into lower-precision representations to reduce the memory footprint and speed up inference. However, in Section 3, we found that when precision is reduced from 32 bits to 8 bits, direct quantization (also called Round-to-Nearest Quantization, RTN-Q) significantly increases the minimum quantization step size, causing activation values below this threshold to be truncated to zero and ultimately rendering the automated circuit discovery method ineffective. Furthermore, RTN-Q reduces activation precision when the model aggregates these values (e.g., in residual streams). When two numbers are added, the larger number dominates the smaller one, masking the activation differences of the selected edge and preventing the algorithm from detecting the influence of the selected edge. This represents the most challenging obstacle in apply-

ing quantization to accelerate automated circuit discovery methods.

These issues stem primarily from the precision degradation of selected edges, and we can use mixed-precision quantization (MPQ) (Saxena et al., 2025; Huang et al., 2025; Lee et al., 2024) to mitigate these effects. Mixed-precision quantization enables selective maintenance of high numerical precision exclusively for specific components under investigation while safely reducing precision elsewhere in the network, avoiding the issues caused by direct quantization. However, most conventional mixed-precision quantization methods require substantial pre-training time to identify important components. Moreover, due to GPU memory limitations, mixed-precision quantization methods typically store high-precision weights in CPU memory, while only the critical weight subset needed for current computation and all low-precision weights are retained on GPU. The dynamic nature of important weights introduces frequent CPU-GPU weight migration in automated circuit discovery, resulting in significant I/O latency.

Our key insight is that circuit discovery and mixed-precision quantization are naturally aligned: since ACDC evaluates one edge at a time, we always know which component needs high precision—the one currently under evaluation. This eliminates MPQ's calibration phase entirely. Moreover, ACDC's sequential evaluation order makes the next high-precision component predictable, enabling us to prefetch its FP32 weights from CPU while the GPU executes low-precision computation, overlapping transfer with compute. Based on this insight, we develop PAHQ (Per-Attention Head Quantization), a training-free method that dynamically assigns FP32 precision to the single attention head under evaluation while running all other components at FP8, coordinated by a three-stream CUDA scheduler that hides transfer latency.

Across three circuit discovery tasks, ACDC accelerated by PAHQ demonstrated over 40% improvement in AUC-ROC for classifying circuit component edges compared to 8-bit RTN quantization. When applied to GPT-2, ACDC accelerated by PAHQ reduces computation time by approximately 80% while achieving memory compression ratios exceeding 30% compared to unaccelerated ACDC.

2 Preliminaries

Circuit Discovery. A transformer model operates as a computational graph (Olah et al., 2020) G, where nodes v correspond to computational components (attention heads and MLP layers), and edges e define activation pathways between components. V and E are the collections of all nodes v and edges e in the graph, respectively. Each node v receives inputs as the sum of outputs from its predecessors $\{u \mid (u \to v) \in E\}$, propagating from input embeddings to the final unembedding layer.

A circuit $C \subseteq G$ is defined as a minimal subgraph that preserves task-critical computation paths from inputs to logits (Olah et al., 2020). To identify such circuits, *activation patching* (Vig et al., 2020; Finlayson et al., 2021) is used to intervene in the computational graph and analyze the contribution of different components. Specifically, given a task distribution $\mathcal{T}$ and input pairs $\mathcal{P} = (\mathbf{z}, \mathbf{z}')$ (clean prompt $\mathbf{z}$ vs. corrupted $\mathbf{z}'$), we construct hybrid activations $E_C(\mathbf{z}, \mathbf{z}')$ by replacing non-circuit edges in $G(\mathbf{z})$ with their counterparts from $G(\mathbf{z}')$. The optimal circuit C^* minimizes the expected loss discrepancy (Zhang et al., 2025):

$$C^* = \arg\min_{C \subseteq G} \mathbb{E}_{\mathcal{P}} |L(E_C(\mathcal{P})) - L(E_G(\mathbf{z}))| \quad (1)$$

where $(\mathbf{z}, \mathbf{z}') \sim \mathcal{T}$, and $L(\mathbf{A})$ measures task-specific loss over logits. To automate this process, ACDC (Conmy et al., 2023) prunes edges with insufficient causal impact:

$$C = G \setminus \{e \in G \mid \Delta L(e) < \tau\}, \quad (2)$$

$$\Delta L(e) = \mathbb{E}_{\mathcal{P}} \left| L(E_{G \setminus e}(\mathcal{P})) - L(E_G(\mathbf{z})) \right|, \quad (3)$$

where τ is a pruning threshold. While ACDC guarantees precision, its $O(|E|)$ complexity becomes prohibitive for large models. EAP (Syed et al., 2023) circumvents this via gradient-based approximation: for an edge $e = (u \to v)$ with clean/corrupted activations $\mathbf{x}_u, \mathbf{x}'_u$, it estimates:

$$L(\mathbf{x}_u) - L(\mathbf{x}'_u) \approx (\mathbf{x}_u - \mathbf{x}'_u) \cdot \frac{\partial L(\mathbf{x}'_u)}{\partial \mathbf{x}_v}, \quad (4)$$

Though EAP reduces the total cost to a single forward-backward pass, its reliance on first-order approximations introduces errors when nonlinear interactions dominate (Hanna et al., 2024b).

Quantization. Quantization (Jacob et al., 2017; Nagel et al., 2019; Gholami et al., 2021) reduces the bit-width of weights and activations to improve

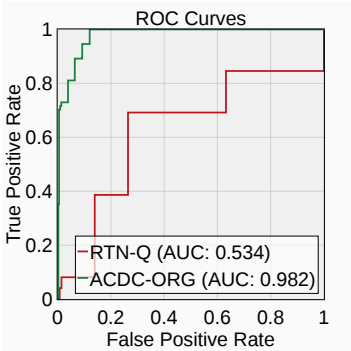

Figure 1: ROC curves. RTN-quantization degrades performance.

computational efficiency, trading off some precision for speed. Consider a group of weights $\mathbf{w}$, the linear operation is $\mathbf{y} = \mathbf{w}\mathbf{x}$, and its corresponding operation after quantization is $\mathbf{y} = Q(\mathbf{w})\mathbf{x}$. Formally, RTN (Round-to-Nearest), the most commonly used quantization method, can be formulated as follows:

$$Q(\mathbf{w}) = \Delta \cdot \text{Round}(\frac{\mathbf{w}}{\Delta}), \quad \Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}} \quad (5)$$

where N is the number of bits and Δ is the quantization scaling factor determined by the absolute maximum value.

3 Direct Quantization Fails in ACDC

Here, we present preliminary studies on directly applying quantization to ACDC to determine whether it can improve ACDC's efficiency.

We directly applied quantization from 32 bits to 8 bits to all weights and activations of the model (Radford et al., 2019) in the ACDC (Conmy et al., 2023) method, which is called Round-to-Nearest Quantization (RTN-Q), on the Indirect Object Identification (IOI) task (Wang et al., 2022) and tested its performance. Figure 1 shows that the AUC-ROC was significantly reduced compared to the original ACDC, indicating that direct quantization is not suitable for it. As mentioned in Section 2, since the replacement operation only changes the activation value of the selected edge, the difference in the model's output is primarily related to the difference in activation values of the selected edge, rather than other components. Specifically, this degradation can be attributed to two main factors: Numerical Underflow and Mantissa Loss.

Numerical Underflow. When the quantization precision of weights and activation values is reduced from 32 bits to 8 bits (or less), the minimum representable precision increases substantially. For FP8_E4M3[1], the minimum quantization step size is: 2^{-6}. This means that if the difference between inference outputs before and after replacing activation values of selected edges is less than the minimum quantization step size, this difference will be truncated to zero. For automated circuit discovery methods, the threshold is likely smaller than this value (Conmy et al., 2023; Syed et al., 2023). Numerical underflow causes the activation value of the selected component or the change in model output to be set to zero when it is smaller than the minimum quantization step size. This causes the automated circuit discovery method to fail in pruning the edge.

Mantissa Loss. RTN quantization reduces numerical precision when activation values are summed (such as in residual connections). When two floating-point numbers are added, the exponents are aligned to the larger number. The mantissa of the smaller number must be right-shifted and discarded until its exponent matches the larger one. Therefore, in the FP8_E4M3 format, which has only 3 mantissa bits, if the exponent difference is 3 or greater (Kuzmin et al., 2024), the significant bits of the smaller number are completely lost. It causes the activation difference introduced by the selected edge to be overwhelmed when aggregated with other activation values, preventing the algorithm from detecting the edge's influence on the model output.

4 Method

To address the precision degradation and computational bottleneck challenges identified in Section 3, we introduce PAHQ (Per-Attention Head Quantization), a training-free mixed-precision quantization method specifically designed for automated circuit discovery methods.

4.1 Per Attention Head Quantization

Key Insight: Natural Alignment Between Circuit Discovery and Mixed-Precision As analyzed in Section 3, ACDC's edge importance criterion depends primarily on the precision of the selected edge's activation values. ACDC evaluates edge importance through the loss difference $\Delta L(e)$ defined in (3). The critical observation is

[1] FP8_E4M3 is an 8-bit floating point storage format, with 4 bits for the exponent, 3 bits for the mantissa, and one bit for the sign.

that this loss difference is dominated by the numerical precision of activations flowing through the edge $e = (u \to v)$ under evaluation, where v and u are nodes in the computational graph G corresponding to computational components. Specifically, let $\mathbf{a}_u^{(high)}$ and $\mathbf{a}_u^{(low)}$ denote the high-precision and low-precision activations from source node u, respectively. The precision-induced error in edge evaluation can be formalized as:

$$\epsilon_{\text{precision}}(e) \triangleq \left| \Delta L(e; \mathbf{a}_u^{high}) - \Delta L(e; \mathbf{a}_u^{low}) \right| \quad (6)$$

Both numerical underflow and mantissa loss issues affect circuit discovery by introducing non-negligible $\epsilon_{precision}(e)$, while other components remain largely unaffected since their activations are not subject to intervention during edge e's evaluation. Our approach leverages a fundamental insight: there exists a natural alignment between activation patching in circuit discovery and mixed-precision quantization strategies. Mixed-precision quantization enables selective maintenance of high numerical precision exclusively for specific components under investigation while safely reducing precision elsewhere in the network, avoiding the issues caused by direct quantization.

Mixed-precision quantization requires extensive calibration to identify important components, which contradicts the goal of accelerating circuit discovery. However, in automated circuit discovery, we inherently know which components are critical—precisely those edges currently under evaluation. This enables us to eliminate the pre-training phase entirely by directly using the selected edges to guide precision allocation.

Selective Precision Allocation Strategy Based on this insight, we propose a dynamic selective precision allocation strategy that adapts to the temporal evaluation sequence. Given the computational graph $G = (V, E)$ from Section 2, we formalize the precision assignment process as follows.

For any edge evaluation at time step t, let $e_t = (u_t \to v_t) \in E$ denote the edge currently under evaluation by ACDC. We define a time-dependent precision allocation function $P_t : V \to \{\text{FP8}, \text{FP32}\}$ for the attention components that dynamically assigns precision levels based on the current evaluation context:

$$P_t(v) = \begin{cases} \text{FP32}, \text{if } v = u_t \\ \text{FP8}, \text{otherwise} \end{cases} \quad (7)$$

For non-attention components (e.g., FFN layers), we uniformly adopt a bfloat16 quantization setting. This strategy ensures non-critical components operate at reduced precision, thereby preserving the faithfulness of $\Delta L(e_t)$ computation while saving GPU memory.

4.2 Algorithm Implementation in Practice

Hierarchical Weight Scheduling Formulation. The dynamic precision allocation requires managing weights across CPU and GPU memory. We store FP32 copies of all attention head weights in CPU memory while keeping their FP8 counterparts on GPU. At each evaluation step t, the conventional approach transfers the required FP32 weights synchronously from CPU to GPU before computation can proceed, creating a serial bottleneck: the GPU idles during every transfer.

However, since ACDC evaluates edges sequentially as $e_1, e_2, \ldots, e_{|E|}$, we can predict which weights will be needed next and exploit this for **predictive scheduling**. Let $S_t = v \mid P_t(v) = \text{FP32}$ denote the set of nodes requiring high precision at step t. While the GPU executes the forward pass at step t using S_t, our scheduler asynchronously prefetches the FP32 weights for S_{t+1} from CPU. This overlaps data transfer with computation, converting the serial bottleneck into a parallel pipeline.

Mixed-Precision Computation Design. Although our method aims to compute only the selected attention head at FP32 while keeping the rest at FP8, GPU hardware prevents directly mixing precisions within a single matrix multiplication kernel, as cuBLAS requires both operands to share the same data type (Micikevicius et al., 2018). To address this limitation, we implement a two-phase computation strategy. For a transformer layer l with H attention heads and a target head $h^* \in \{1, 2, \ldots, H\}$, we define the attention computation process as follows.

First, we compute projections separately for each precision level. For each attention component $comp \in \{Q, K, V\}$, we generate low-precision outputs over all H heads and a high-precision output for the target head h^* only:

$$\mathbf{A}_{comp,8}^{(l)} = f_{FP8}(\mathbf{T}, \mathbf{W}_{comp,8}^{(l)}) \quad (8)$$

$$\mathbf{A}_{comp,32}^{(l,h^*)} = f_{FP32}(\mathbf{T}, \mathbf{W}_{comp,32}^{(l,h^*)}) \quad (9)$$

where $\mathbf{T} \in \mathbb{R}^{B \times S \times d}$ is the input tensor, B denotes batch size, S sequence length, d_k the head dimen-

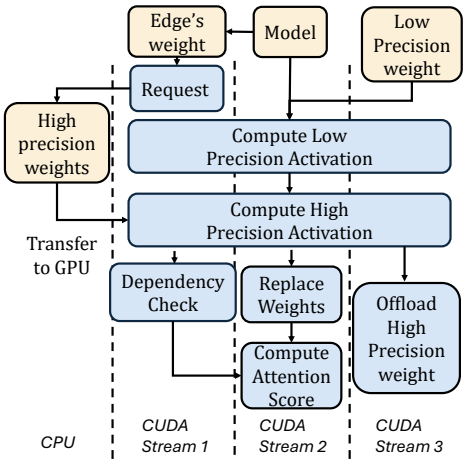

Figure 2: Three-Stream Parallel Scheduler. We overlap high-precision weight transfers with low-precision computations to mask transfer latency.

sion, and f_{prec} attention projection at precision $prec \in \{\text{FP8}, \text{FP32}\}$.

Next, we perform selective replacement to construct the mixed-precision attention tensor. We define the MixedAssembly operation for each head h as:

$$\mathbf{A}_{comp}^{(l,h)} \triangleq \begin{cases} \mathbf{A}_{comp,32}^{(l,h)}, & h = h^* \\ \text{Cast}_{FP32}\left(\mathbf{A}_{comp,8}^{(l,h)}\right), & h \neq h^* \end{cases} \quad (10)$$

where Cast_{FP32} converts tensors to FP32 format.

Finally, we unify all activations to FP32 precision to ensure consistency for subsequent matrix operations:

$$\mathbf{A}_{comp}^{(l)} = \text{Concat}_{head}([\mathbf{A}_{comp}^{(l,1)}, \ldots, \mathbf{A}_{comp}^{(l,H)}]) \quad (11)$$

This approach maintains the numerical precision required for the critical attention head while achieving memory savings through reduced precision for non-critical components.

Scheduler Implementation. To optimize performance, we design a three-stream parallel scheduler that overlaps computation and communication. Let $\mathcal{S} = \{S_{load}, S_{low}, S_{high}\}$ denote the three CUDA streams for weight loading, low-precision computation, and high-precision computation, respectively.

The scheduling process implements the following pipeline for each layer l and target head h^*:

Phase 1: Asynchronous Weight Prefetching We initiate asynchronous weight transfers from CPU to GPU for both the target attention head and output projection:

$$S_{load} : \mathbf{W}_{QKV,32}^{(l,h^*)}, \mathbf{W}_{O,32}^{(l)} \xrightarrow{async} \text{GPU}, \quad (12)$$

where $\mathbf{W}_{QKV,32}^{(l,h^*)}, \mathbf{W}_{O,32}^{(l)}$[2] represents the combined weight matrices for Query, Key, Value and Output projections of the target attention head h^* in layer l stored in FP32 precision.

Phase 2: Overlapped Computation For each attention component $comp \in \{Q, K, V\}$, we execute the following operations:

$$S_{low} : \text{compute } \mathbf{A}_{comp,8}^{(l)} \text{ (Eq. 8, async)} \quad (13)$$

$$\text{Sync}(S_{load}, \mathbf{W}_{comp,32}^{(l,h^*)})$$

$$S_{high} : \text{compute } \mathbf{A}_{comp,32}^{(l,h^*)} \text{ (Eq. 9, async)} \quad (14)$$

where $Sync(\cdot)$ is a stream synchronization operation that waits for the specified stream or memory transfer to complete before proceeding to the next operation. These operations reduce idle time by overlapping computations and memory transfers.

Phase 3: Synchronization and Assembly We perform the attention computation as follows:

$$\text{Sync}(S_{low}, S_{high})$$

$$\mathbf{A}_{comp}^{(l)} \overset{(10)}{=} \mathcal{M}_{h^*}(\mathbf{A}_{comp,8}^{(l)}, \mathbf{A}_{comp,32}^{(l,h^*)}, h^*) \quad (15)$$

$$\text{Sync}(S_{load}, \mathbf{W}_{O,32}^{(l)})$$

$$\mathbf{Y} = \text{Attention}(\mathbf{A}_Q^{(l)}, \mathbf{A}_K^{(l)}, \mathbf{A}_V^{(l)}) \cdot \mathbf{W}_{O,32}^{(l)}, \quad (16)$$

Let T_{tr}, T_l, T_h, T_a denote transfer, low-precision computation, high-precision computation, and assembly time, respectively. Since the CPU→GPU transfer on S_{load} runs concurrently with low-precision computation on S_{low}, the total time per layer is approximately:

$$T_{\text{total}} \approx \max(T_{tr}, T_l) + T_h + T_a \quad (17)$$

In practice, $T_{tr} > T_l$ for typical model sizes, so $\max(T_{tr}, T_l) \approx T_{tr}$ and the low-precision computation is effectively performed at no additional cost. Compared to a sequential baseline that pays $T_{tr} + T_l + T_h + T_a$ in full, our overlapping strategy eliminates $\min(T_{tr}, T_l)$ from the critical path. The complete PAHQ-accelerated ACDC algorithm and schematic diagram are provided in Appendix E.

5 Experiment

5.1 Experimental Setting

Tasks. We selected three tasks that are commonly studied in automatic circuit discovery: Indirect Ob-

[2] Since the output projection operates on the concatenated activations from all heads, we compute it at FP32 to avoid introducing quantization error into the assembled output.

ject Identification (IOI) (Wang et al., 2022), Docstring (Heimersheim and Janiak, 2023) and Greater Than (Nanda et al., 2023). For task introduction, please refer to Appendix A.

Baselines. For baselines, we employ ACDC (Conmy et al., 2023) as our baseline method for quantization and compare the performance of HISP (Michel et al., 2019), SP (Cao et al., 2021), EAP (Syed et al., 2023), Edge Pruning (Bhaskar et al., 2024), RTN quantization on ACDC, original ACDC, and our method. We present the comparison results with Edge Pruning in Appendix C.

Metrics. Since we apply quantization to the ACDC method in this work, we adopt its original evaluation metric (the Area Under the Receiver Operating Characteristic curve, AUC-ROC) to assess faithfulness. For a more detailed description of metrics, please refer to Appendix A. We also assess faithfulness using the metric proposed by (Hanna et al., 2024b) in Appendix A.

Models. The models used are the same as ACDC, including gpt2 (Radford et al., 2019), attn-4l (Heimersheim and Janiak, 2023), and redwood-2l (Gokaslan and Cohen, 2019). All of these models are downloaded from HuggingFace.

5.2 Experimental Results

This section compares the performance of different methods in terms of our primary metrics: faithfulness, task performance, runtime, and memory requirements.

Method	AUC-ROC					
	IOI		Greater Than		Docstring	
	KL	Task	KL	Task	KL	Task
ACDC	**0.98**	**0.97**	**0.91**	<u>0.87</u>	**0.95**	**0.94**
RTN-Q	0.53	0.58	0.62	0.50	0.47	0.53
HISP	0.78	0.83	0.70	0.68	0.81	0.82
SP	0.82	0.72	0.80	0.81	0.83	0.81
EAP	0.71	0.70	0.83	**0.89**	0.83	0.83
PAHQ	<u>0.96</u>	<u>0.94</u>	<u>0.87</u>	0.84	<u>0.89</u>	<u>0.89</u>

Table 1: AUC-ROC for different circuit discovery methods on specified tasks and models. The thresholds used were logarithmically evenly distributed between 0.001 and 3.16, for a total of 21 values, consistent with ACDC. "**bold**" and "<u>underlined</u>" represents the best and second best performance. RTN-Q and PAHQ are abbreviations of RTN-Q-accelerated ACDC and PAHQ-accelerated ACDC.

AUC-ROC Performance Analysis. As illustrated in Tab. 1, PAHQ-accelerated ACDC outperforms both RTN-Q ACDC and EAP approaches across three tasks, and only has minimal faithfulness loss compared to the original ACDC method. For the IOI task, PAHQ-accelerated ACDC achieves AUC-ROC values of 0.96 and 0.94 when optimized using KL divergence and task metrics, substantially exceeding RTN-Q ACDC and EAP. However, on the Greater Than task, our method slightly underperforms EAP (Syed et al., 2023) due to the lower performance ceiling imposed by the original ACDC. Additionally, we observe that EAP performs poorly on IOI, which is the same as the results in (Hanna et al., 2024b).

Accuracy Across Different Thresholds and Models Tab. 2 presents a detailed performance comparison of various quantization methods, tasks, thresholds, and model configurations. Our method consistently outperforms RTN-Q while only suffering a small performance loss over unaccelerated ACDC. E.g., in the IOI task, when the threshold of the gpt2 is 0.001, PAHQ-accelerated ACDC achieves 0.51 on KL divergence and 0.52 on task metrics, significantly outperforming RTN-Q ACDC while only slightly declining from the original ACDC method. This trend remains consistent across different model sizes (Attn-4l and Redwood-2l). Even on the harder task of Docstring, our method still performs well. We also assess our method's scalability to larger models in Appendix B.

Runtime and Memory. To demonstrate the improvement of the circuit discovery method using PAHQ and its performance relative to the RTN-Q quantization method, we tested the operating results in three cases, as shown in Tab. 3. Our method, through lower-level stream scheduling and matrix multiplication optimization, reduces about 80% running time and nearly 1/3 memory usage compared to the original method, close to RTN-Q ACDC. This is because our method only needs to store one additional high-precision attention head compared to the RTN-Q method.

5.3 Ablation Studies

Scheduling. Tab. 4 shows adding a scheduler can significantly reduce the running time. When both the weight loading stream and the low/high precision calculation stream are enabled, the running time is only 20 minutes, which is about 79% less than the worst case. Enabling only the weight loading stream (49 minutes) or the low/high precision

Threshold	Method	Metric	Models and Tasks								
			IOI			Greater Than			Docstring		
			gpt2	4l	2l	gpt2	4l	2l	gpt2	4l	2l
0.001	ACDC	KL div	**0.54**	**0.45**	**0.37**	**0.60**	**0.53**	**0.48**	**0.16**	**0.06**	0.03
		Task	**0.54**	**0.42**	**0.35**	**0.64**	**0.49**	**0.45**	**0.17**	**0.08**	**0.07**
	RTN-Q ACDC	KL div	0.32	0.28	0.20	0.43	0.30	0.39	0.08	0.04	0.01
		Task	0.33	0.29	0.19	0.41	0.35	0.35	0.10	0.05	0.02
	PAHQ ACDC	KL div	0.51	0.44	0.35	0.58	0.48	0.46	**0.16**	0.04	**0.04**
		Task	0.52	0.39	0.29	0.61	0.47	0.43	0.14	0.06	0.04
0.01	ACDC	KL div	**0.57**	**0.50**	**0.41**	**0.60**	**0.55**	0.38	0.14	**0.09**	**0.06**
		Task	**0.53**	**0.46**	0.41	**0.65**	**0.52**	**0.43**	**0.15**	**0.07**	**0.04**
	RTN-Q ACDC	KL div	0.33	0.26	0.22	0.42	0.37	**0.41**	0.11	0.08	0.03
		Task	0.36	0.25	0.17	0.49	0.39	0.36	0.09	0.04	0.00
	PAHQ ACDC	KL div	0.56	0.49	0.39	0.58	**0.55**	0.36	**0.15**	0.06	0.04
		Task	0.51	**0.46**	**0.43**	0.60	0.48	0.39	0.12	**0.07**	**0.04**
0.1	ACDC	KL div	**0.60**	**0.51**	0.44	**0.65**	**0.49**	**0.55**	**0.14**	**0.10**	**0.08**
		Task	**0.58**	**0.53**	**0.43**	**0.65**	**0.54**	**0.49**	**0.14**	**0.06**	0.03
	RTN-Q ACDC	KL div	0.32	0.29	0.24	0.50	0.46	0.41	0.08	0.05	0.03
		Task	0.32	0.28	0.25	0.43	0.31	0.39	0.10	0.04	0.01
	PAHQ ACDC	KL div	0.54	0.50	**0.45**	0.63	0.48	0.45	**0.14**	0.09	0.06
		Task	0.56	0.48	0.40	0.57	0.51	0.42	0.12	**0.06**	**0.04**

Table 2: Task accuracy for different quantization methods across various tasks, thresholds, and models. "Task" in column "metric" means task metric. Attn-4l and Redwood-2l are represented as 2l and 4l. For different metrics and tasks, within each threshold group, the highest value in each column is bolded. When there are no tied highest values, the second highest value is underlined.

Method	Performance Comparison					
	gpt-2		attn-4l		redwood-2l	
	Time (m:s)	Mem (GB)	Time (m:s)	Mem (GB)	Time (m:s)	Mem (GB)
ACDC	99:18	6.23	4:03	3.30	0:28	1.54
RTN-Q	27:47	**4.15**	1:17	**1.96**	0:09	**0.99**
PAHQ	**20:36**	4.24	**1:01**	2.11	**0:06**	1.21

Table 3: Runtime and memory comparison of quantization methods on the IOI task (threshold = 0.001). Bold and underlined values represent the best and second-best results. Time reported in minutes:seconds format.

Weight Loading Stream	Low/High-Precision Computation Stream	Runtime (m)
✓	✓	**20**
✓	✗	49
✗	✓	72
✗	✗	94

Table 4: Impact of scheduler on runtime. Experiments were conducted using the GPT-2 model on the IOI task.

calculation stream (72 minutes) can also improve parts of performance. This is expected: without the dedicated loading stream, the GPU must wait for synchronous weight transfers that exceed the high-precision computation time, making transfer the dominant bottleneck.

Different Quantization Precisions. From Tab. 5, it is evident that different quantization precisions significantly impact model performance. 4-bit quantization severely compromises model accuracy and faithfulness. This occurs because, although the selected edge's activation is computed at full precision, the subsequent layers operate at only 4 bits, which cannot preserve the small numerical differences that ACDC relies on for edge evaluation. Therefore, it is almost impossible to use 4-bit quantization. In contrast, 16-bit quantization offers minimal precision improvements compared to 8-bit. To validate the correctness of our quantization method, we tested the performance of the model under different quantization scales. Please refer to Appendix D for details.

Precision	Accuracy	AUC-ROC
4-bit	0.06	0.61
8-bit	0.54	0.93
16-bit	0.55	0.96

Table 5: Impact of different quantization precisions on results. Experiments were conducted with a threshold of 0.001 on the IOI task using the gpt-2 model.

6 Related Work

Circuit Discovery. Neural networks can be abstracted as computational graphs, with circuits serving as the fundamental computational units and building blocks (Olah et al., 2020). Circuits are subgraphs representing critical components necessary for specific tasks, and circuit discovery tasks utilize task information and network structure (Bereska and Gavves, 2024) to identify important operations and attribute outputs to specific components (Miller et al., 2024), thereby extracting the model's critical circuits for specific tasks. Existing research has found that this attribution approach can significantly enhance interpretability for designated tasks in small models, such as knowledge-intensive tasks (Meng et al., 2022), grammatical functions (Wang et al., 2022), temporal reasoning tasks (Hanna et al., 2024a), and others.

However, extending circuit discovery methods to large models remains challenging due to the complexity of causal interventions and the exponential growth of circuit interactions as the number of components increases. Although Automated Circuit Discovery (ACDC) (Conmy et al., 2023) automates circuit discovery and ensures high faithfulness through exhaustive recursive activation patching, its $O(|E|)$ complexity with respect to model components becomes prohibitive for large-scale networks. To address this issue, Edge Attribution Patching (EAP) (Syed et al., 2023) achieves linear efficiency via gradient-based edge estimation, but this comes at the cost of approximation errors in multi-layer nonlinear computations. EAP-IG (Hanna et al., 2024b) proposed a new faithfulness metric, and improves edge attribution patching's faithfulness by integrated gradients. Edge Pruning (Bhaskar et al., 2024) prunes by training the model's edge mask, improving faithfulness at the cost of time. Our method balances the precision-efficiency trade-off by introducing per-attention-head quantization and optimizing the inference process, thereby maintaining high precision while reducing computational costs.

Quantization. Quantization is a method of converting high-precision model parameters into low-precision representations, which can significantly reduce model size and accelerate inference (Gholami et al., 2021). However, precision requirements exhibit significant heterogeneity across diverse tasks (Nagel et al., 2021; Holavanahali, 2023), presenting challenges in determining optimal bit allocations universally. To address this problem, (Lin et al., 2024) proposed a quantization framework that selectively preserves critical dimensions; SmoothQuant achieves efficient quantization via channel-wise rescaling operations (Xiao et al., 2023). These approaches require additional hyperparameters that need calibration across datasets and models, imposing substantial computational overhead. To address these limitations, (Yao et al., 2022) introduced a zero-shot quantization paradigm without fine-tuning procedures. However, it fails to maintain the interpretability of internal model mechanisms, and circuit discovery tasks fundamentally rely on understanding and preserving these intricate mechanisms.

Inspired by task-specific quantization techniques (Frantar and Alistarh, 2023; Yuan et al., 2023; Lee et al., 2024), we propose PAHQ, which differs from these methods by exploiting the sequential, single-edge evaluation structure of circuit discovery to guide precision allocation dynamically—eliminating the need for calibration data or additional training while preserving the activation fidelity required for faithful circuit identification.

7 Conclusions

We introduce PAHQ (Per Attention Head Quantization), an efficient quantization scheme for automated circuit discovery methods. PAHQ leverages the intrinsic characteristics of automated circuit discovery by selectively maintaining high-precision representations of critical attention heads, thereby significantly reducing computational overhead while preserving circuit identification faithfulness. This method requires no additional training and can be used as a plug-and-play solution. Experimental results demonstrate that PAHQ achieves an average reduction of approximately 80% in time consumption and 30% in memory usage when implemented within the ACDC framework with only slight performance loss, providing an acceleration approach for mechanistic interpretability research without sacrificing analytical quality.

Limitations

Applicability to Circuit Discovery Algorithms Our proposed method, PAHQ, is designed to accelerate circuit discovery techniques that rely on the precise, sequential evaluation of individual edges within the computational graph. Its effectiveness is most pronounced when applied to algorithms like ACDC, where only one edge's activations are patched per inference pass. This allows for a highly targeted application of high-precision computation, maximizing efficiency gains. However, the scope of PAHQ's applicability does not extend uniformly to all circuit discovery paradigms. For methods such as Edge Attribution Patching (EAP), which approximate ACDC by evaluating the influence of multiple edges within a layer simultaneously during a single backward pass, the benefits of our approach would be diminished. To maintain faithfulness, such methods would require holding all concurrently evaluated edges and their associated components at high precision, which would significantly reduce the promised improvements in memory consumption and runtime.

Scalability to Large-Scale Models This work demonstrates significant performance improvements on models commonly used in mechanistic interpretability research, such as the GPT-2 family. It is important to acknowledge that a shared limitation across most current circuit discovery methods is their primary application to these relatively small-scale models. While PAHQ serves as an effective accelerator, it primarily mitigates the exponential growth in computational cost associated with the underlying discovery algorithms as model size increases. It does not, however, fundamentally solve the scalability challenge. The core problem of applying exhaustive, intervention-based analyses to foundation models with tens or hundreds of billions of parameters remains an open and critical research area. Our method makes the application to moderately larger models more feasible but does not yet provide a complete solution for today's largest models.

Reliance on Empirical Validation The justification for our mixed-precision strategy, particularly regarding its ability to preserve the fidelity of patched activations through subsequent computations, is primarily supported by strong empirical results. We demonstrate through extensive experiments that maintaining high precision for a single critical component is sufficient for the ACDC algorithm to function correctly. However, the paper provides limited theoretical analysis on how the high-precision signal avoids degradation as it propagates through downstream layers operating at lower precision. Furthermore, the choice to configure non-attention components with `bfloat16` (bf16) precision was determined empirically through experimental testing, rather than being derived from a formal theoretical analysis. A more profound analytical investigation into the distinct precision requirements of different component types (e.g., attention vs. MLP layers) could yield further insights and potentially lead to more refined optimization strategies.

References

Lukas Bereska and Efstratios Gavves. 2024. Mechanistic interpretability for ai safety–a review. *arXiv preprint arXiv:2404.14082*.

Adithya Bhaskar, Alexander Wettig, Dan Friedman, and Danqi Chen. 2024. Finding transformer circuits with edge pruning. *Advances in Neural Information Processing Systems*, 37:18506–18534.

Steven Cao, Victor Sanh, and Alexander M Rush. 2021. Low-complexity probing via finding subnetworks. *arXiv preprint arXiv:2104.03514*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Arthur Conmy, Augustine N. Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. 2023. Towards automated circuit discovery for mechanistic interpretability. *arXiv preprint arXiv:2304.14997*.

Tom Fawcett. 2006. Introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874.

Mark Finlayson, Aaron Mueller, Sebastian Gehrmann, and 1 others. 2021. Causal analysis of syntactic agreement mechanisms in neural language models. *arXiv preprint arXiv:2106.06087*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. *Preprint*, arXiv:2301.00774.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *Preprint*, arXiv:1510.00149.

M. Hanna, O. Liu, and A. Variengien. 2024a. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. In *Advances in Neural Information Processing Systems*, volume 36. NeurIPS.

Michael Hanna, Sandro Pezzelle, and Yonatan Belinkov. 2024b. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. *Preprint*, arXiv:2403.17806.

Stefan Heimersheim and Jett Janiak. 2023. A circuit for python docstrings in a 4-layer attention-only transformer. *Alignment Forum*. Online; accessed March 2025.

Anusha Holavanahali. 2023. *An Analysis of Heterogeneous Quantization Schemes for Neural Networks*. Rochester Institute of Technology.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and Xiaojuan Qi. 2025. Slim-llm: Salience-driven mixed-precision quantization for large language models. *Preprint*, arXiv:2405.14917.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *arXiv preprint arXiv:1712.05877*.

Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *Preprint*, arXiv:1806.08342.

Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Andrey Kuzmin, Mart Van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters, and Tijmen Blankevoort. 2024. Fp8 quantization: The power of the exponent. *Preprint*, arXiv:2208.09225.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2024. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. *Preprint*, arXiv:2306.02272.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for llm compression and acceleration. *Preprint*, arXiv:2306.00978.

Omid Nejati Manzari, Hamid Ahmadabadi, Hossein Kashiani, Shahriar B. Shokouhi, and Ahmad Ayatollahi. 2023. Medvit: A robust vision transformer for generalized medical image classification. *Computers in Biology and Medicine*, 157:106791.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed precision training. *Preprint*, arXiv:1710.03740.

Joseph Miller, Bilal Chughtai, and William Saunders. 2024. Transformer circuit faithfulness metrics are not robust. *Preprint*, arXiv:2407.08734.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2025. Large language models: A survey. *Preprint*, arXiv:2402.06196.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization. *Preprint*, arXiv:2106.08295.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. 2019. Data-free quantization through weight equalization and bias correction. *arXiv preprint arXiv:1906.04721*.

Neel Nanda and Joseph Bloom. 2022. Transformerlens. https://github.com/TransformerLensOrg/TransformerLens.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. 2023. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. 2020. Zoom in: An introduction to circuits. *Distill*. Https://distill.pub/2020/circuits/zoom-in.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Utkarsh Saxena, Sayeh Sharify, Kaushik Roy, and Xin Wang. 2025. Resq: Mixed-precision quantization of large language models with low-rank residuals. *Preprint*, arXiv:2412.14363.

Aaquib Syed, Can Rager, and Arthur Conmy. 2023. Attribution patching outperforms automated circuit discovery. In *NeurIPS Workshop on Attributing Model Behavior at Scale*.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. 2020. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2022. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *Preprint*, arXiv:2211.00593.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. 2023. Rptq: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*.

Lin Zhang, Wenshuo Dong, Zhuoran Zhang, Shu Yang, Lijie Hu, Ninghao Liu, Pan Zhou, and Di Wang. 2025. Eap-gp: Mitigating saturation effect in gradient-based automated circuit identification. *arXiv preprint arXiv:2502.06852*.

A Experiment Details

A.1 Environment

All our experiments were conducted on a server equipped with NVIDIA H20-NVLink GPU (96GB) and 16 vCPU AMD EPYC 9K84 96-Core Processor. For low-precision operations, we utilize the FP8_E4M3 format.

A.2 Tasks

We selected 3 tasks from the ACDC paper: Indirect Object Identification (IOI), Docstring, and Greater Than:

- **IOI** (Wang et al., 2022): This task tests a model's ability to identify the indirect object in sentences like "When Mary and John went to the store, John gave a gift to Mary." The model must correctly identify "Mary" as the recipient of the gift. The dataset consists of 1,000 examples with varying names and contexts, balanced across different sentence structures.

- **Docstring** (Heimersheim and Janiak, 2023): This task evaluates a model's ability to generate appropriate Python function documentation from function signatures and implementations. The dataset comprises 10,000 Python functions from diverse libraries, with varying complexity and domain specificity.

- **Greater Than** (Nanda et al., 2023): This task evaluates a model's numerical reasoning by asking it to determine which of two numbers is larger. The dataset includes 2,000 pairs of numbers with varying digit lengths and magnitudes, testing both simple and complex numerical comparisons.

Since the original ACDC paper provides datasets in non-natural language token form, which are not interoperable between models with different tokenizers, we first convert all data to natural language form and then transform them using the respective model's tokenizer.

A.3 Metrics

For faithfulness metrics across all three tasks, we follow the approach in the ACDC paper by scanning across a range of ACDC thresholds τ, SP regularization parameters λ, or the number of pruned HISP elements k. We plot pessimistic line

segments between points on the Pareto frontier of TPR and FPR, covering this range of thresholds (Fawcett, 2006).

We employ the Area Under the Receiver Operating Characteristic curve(AUC-ROC) to evaluate faithfulness, as high true positive rates (TPR) and low false positive rates (FPR) conceptually correspond to the method's ability to identify subgraphs corresponding to the underlying algorithm implemented by the neural network and the method's ability to avoid including components that do not participate in eliciting the behavior, respectively. Following the original paper, we also evaluate the performance of these methods in terms of task accuracy, faithfulness, runtime, and memory footprint when using either KL divergence (Kullback and Leibler, 1951) or task-specific metrics as the loss function.

Faithfulness Metric Michael Hanna et al. (Hanna et al., 2024b) pointed out that in the moderate overlap range (approximately 0.5–0.6), different circuits exhibit widely varying faithfulness, and overlap alone cannot predict model behavior; even in single-task settings, circuits whose node overlap with the human benchmark reaches 90% can have faithfulness as low as 0% or only 51%, indicating that high overlap does not necessarily correspond to high faithfulness.

They attribute this phenomenon to the fact that overlap metrics ignore critical components: even if the overall overlap is high, omission of important nodes or edges can cause faithfulness to collapse; moreover, overlap metrics are inherently symmetric and cannot capture the asymmetric nature of faithfulness. Even adopting asymmetric overlap measures such as recall still fails to reliably predict actual faithfulness. Consequently, in addition to using the metrics from the original paper, we also employ the faithfulness evaluation metric proposed in (Hanna et al., 2024b) to comprehensively assess our quantized method. The result is shown in Tab 6:

As shown in table 6, after quantization by our method, the performance of ACDC only decreases slightly, while the results after quantization by RTN-Q decrease significantly.

A.4 Task Metrics

For detailed definitions of task metrics and comparative studies with other metrics (such as KL divergence, performance matching, and methods based on performance changes), please refer to Appendices C and E.2 of (Conmy et al., 2023). These sections thoroughly discuss the advantages and disadvantages of various metrics, and why KL divergence might be superior to logit difference in most of cases.

In this research, we employed logit difference as the sole task metric for our optimization process. This metric calculates the difference in logit values between the correct answer and distractors in specific tasks.

	Task		
Method	**IOI**	**Docstring**	**Greater-Than**
ACDC	0.92	0.90	0.95
RTN-Q	0.27	0.31	0.40
PAHQ	0.85	0.84	0.91

Table 6: The faithfulness result. To value the faithfulness fairly, we used the metric proposed by (Hanna et al., 2024b). All of experiments run with threshold=0.01. RTN-Q and PAHQ are the abbreviations of RTN-Q-accelerated ACDC and PAHQ-accelerated ACDC.

A.5 Edge Cutting Process

As shown in Figure 3, the left graph displays the line chart of circuit edge count changes with step size in the original ACDC algorithm, while the right graph shows the results after applying our quantization method. From the comparison, it is clear that both methods exhibit very similar trends during the circuit pruning process. Both graphs start with approximately 35,000 edges, and as the number of steps increases, the edge count decreases in a similar pattern. Notably, although our method increases the total number of steps to about 700 (compared to 500 steps in the original method), the overall sparsification pattern remains consistent, indicating that the quantization process does not significantly alter the sparsity characteristics or pruning efficiency of ACDC. This demonstrates that our quantization method maintains the core performance of the original algorithm while only slightly affecting the computational process, and as previously mentioned, still far exceeds the original algorithm in terms of speed. This minor increase in step count is an acceptable trade-off, as it results in significant performance improvements.

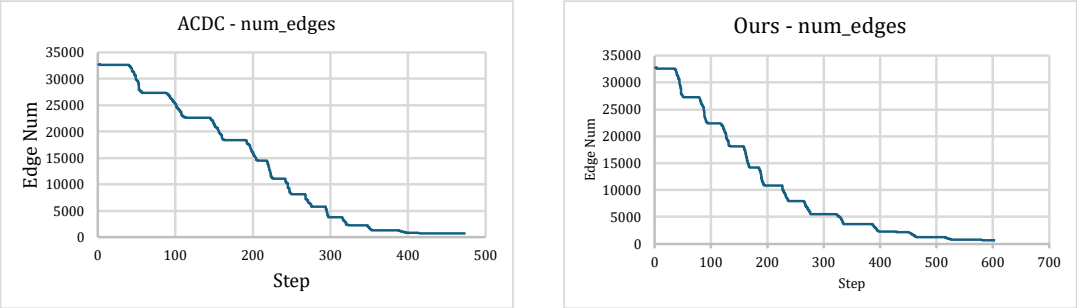

Figure 3: A line graph showing the change in the number of edges in the ACDC circuit before and after quantization using our method as a function of step size. The graph demonstrates that after applying our quantization method, the ACDC circuit does not experience significant changes in terms of sparsity or pruning speed. Although the number of algorithm steps slightly increases, our method still far exceeds the original algorithm in terms of speed.

B Scalability to Larger Models

Our main experimental results focus on GPT-2-small, which is the standard benchmark used in the majority of circuit discovery literature (Hanna et al., 2024b; Syed et al., 2023; Bhaskar et al., 2024) for computational efficiency and fair comparison with existing methods. To demonstrate PAHQ's scalability, we conducted additional experiments on larger GPT-2 variants.

B.1 Experimental Setup

We evaluated PAHQ on GPT-2-medium, GPT-2-large, and GPT-2-XL using the IOI task with a threshold of 0.01. Due to computational constraints on larger models, we implemented a batched version of edge evaluation. All experiments were conducted on NVIDIA H20. For reference, ACDC achieves a KL divergence of 0.79 on the IOI task with GPT-2-small.

B.2 Results

Tab. 7 presents the performance of PAHQ across different model scales, compared with the Edge Attribution Patching (EAP) baseline (Syed et al., 2023).

Model	Batch Size	KL Div (PAHQ)	KL Div (EAP)
GPT-2-medium	6	0.85	3.20
GPT-2-large	5	0.93	3.64
GPT-2-XL	4	0.88	3.92

Table 7: Performance comparison on larger GPT-2 variants. Lower KL divergence indicates better performance.

The results demonstrate that PAHQ scales effectively to larger transformer models while maintaining competitive performance. Across all model scales, PAHQ significantly outperforms the EAP baseline. Notably, on all of these models, PAHQ achieves an average KL divergence of 0.89, which represents only a modest degradation compared to ACDC's performance on GPT-2-small (0.79), while providing substantial computational speedup.

The original ACDC method becomes computationally prohibitive on larger models, even with batch processing optimizations; therefore, we are unable to provide ACDC results on larger models.

C Comparison with Edge Pruning

This appendix provides a detailed comparison between PAHQ and Edge Pruning (Bhaskar et al., 2024), addressing the relationship between training steps, dataset size, and performance in circuit discovery methods.

C.1 Methodological Considerations

An important consideration regarding Edge Pruning's implementation concerns the relationship between training steps and dataset size. The original Edge Pruning paper reports that runtime "barely increases" as dataset size grows. However, this occurs because the method uses a fixed number of training steps (3,000) regardless of dataset size, as confirmed in their official implementation[3]

This design choice means that Edge Pruning processes exactly 3,000 mini-batches regardless of whether the training set contains 100, 10,000, or 100,000,000 examples. Consequently, the method does not reflect how Edge Pruning would behave if

[3] See https://github.com/princeton-nlp/Edge-Pruning/blob/main/src/prune/fpt2_ioi.py for details.

allowed to run until convergence on larger datasets. Due to this implementation detail and concerns about fair comparison, we did not include Edge Pruning as a primary baseline in our main experimental results. Instead, we provide this comprehensive analysis to address the relationship between training budget and performance.

C.2 Experimental Setup

We evaluated Edge Pruning under multiple regimes of training-step budgets and dataset sizes to ensure a comprehensive comparison. All experiments were conducted on the same hardware (a single NVIDIA H20 GPU) using the IOI task. Hyperparameters for Edge Pruning matched those in the original paper, with the learning rate maintained at 0.8 to avoid instability issues observed at higher rates.

C.3 Results

Table 8 presents the performance comparison across different dataset sizes and training steps. Each cell shows KL divergence and runtime in the format "KL & Runtime".

Dataset Size	Metric	Training Steps			
		400 Steps	800 Steps	1600 Steps	3000 Steps
200	KL div	3.56	2.38	0.46	0.23
	Time	31 min	57 min	112min	199min
400	KL div	3.83	1.72	0.48	0.27
	Time	29 min	63min	98min	208min
100,000	KL div	4.17	2.41	0.39	0.19
	Time	35 min	52 min	133min	212min
PAHQ ACDC	KL div	**0.83**			
	Time	**22 min**			

Table 8: Performance comparison between Edge Pruning and PAHQ across different dataset sizes and training steps.

A key practical consideration is computational resource usage. PAHQ completes circuit discovery in 22 minutes using only 5 GB of GPU memory, whereas Edge Pruning requires 33 GB of GPU memory and approximately 3.5 hours for comparable performance. This substantial reduction in memory footprint enables PAHQ to run on modest-memory GPUs, making it significantly more accessible in resource-limited environments.

While Edge Pruning can achieve lower KL divergence with sufficient training steps and computational resources, PAHQ offers a compelling trade-off between performance and practical constraints. The modest increase in KL divergence (0.83 vs. 0.19-0.27 for Edge Pruning at 3000 steps) is offset by dramatic improvements in memory efficiency and runtime, making circuit discovery accessible to researchers with limited computational resources.

D Quantization Strategy

This appendix provides a detailed ablation study comparing PAHQ's selective quantization strategy against full-network quantization approaches to demonstrate the effectiveness of our targeted precision preservation.

D.1 Experimental Design

To validate PAHQ's quantization strategy, we conducted a systematic comparison between selective quantization (our approach) and global quantization methods. The experiment follows a two-phase incremental quantization protocol:

Phase 1: Non-critical Component Quantization We begin with the full-precision circuit discovered by ACDC. Following the model's reverse topological order, we iteratively select additional attention heads classified as non-critical and quantize their weights from FP32 to FP8, while preserving all critical components in FP32 precision. We record the overall task accuracy after each incremental quantization step.

Phase 2: Critical Component Quantization After all non-critical attention heads have been quantized to FP8, we systematically quantize the remaining critical attention head weights from FP32 to FP8 in 10% increments until all weights are quantized. At each step, we measure the impact on task performance to understand the sensitivity of critical components to precision reduction.

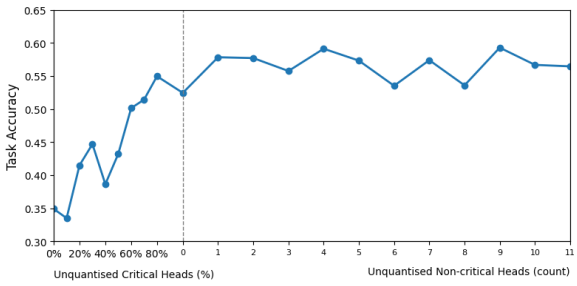

Figure 4: The results of quantization strategy comparison.

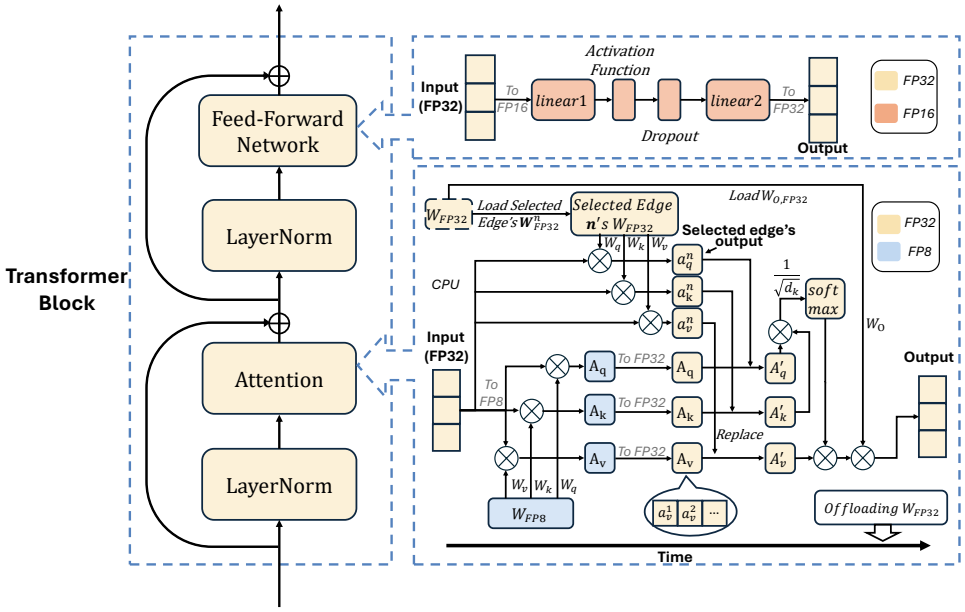

Figure 5: The full algorithm and implementation of PAHQ.

D.2 Results and Analysis

Figure 4 presents the results of our quantization strategy comparison. The experimental results confirm two key findings:

1. A uniform 8-bit quantization approach, which quantizes both critical and non-critical components simultaneously, results in substantial accuracy degradation. This demonstrates that treating all network components equally during quantization is suboptimal for maintaining task performance.

2. PAHQ's targeted preservation of FP32 precision on critical components while quantizing non-critical components to FP8 is essential for maintaining high accuracy under memory-constrained scenarios. This selective approach allows us to achieve significant memory savings without compromising the faithfulness of circuit discovery.

These results provide empirical validation for PAHQ's core design principle: the importance of maintaining high precision for components identified as critical by circuit discovery methods while safely reducing precision elsewhere. This targeted approach enables substantial computational savings without sacrificing the reliability of mechanistic interpretability analysis.

The findings also highlight the limitations of naive quantization approaches in the context of interpretability methods, where the preservation of critical pathway faithfulness is paramount. Our selective quantization strategy offers a principled solution to this challenge, making large-scale circuit discovery more computationally feasible.

E Implementation: Integration with ACDC

In this section, we provide a detailed demonstration of how PAHQ readily integrates with the ACDC method. As mentioned in Preliminary, ACDC is a powerful circuit discovery technique that identifies critical connections within the model by analyzing the impact of activation replacements on downstream computations. The algorithm below describes the implementation details of the PAHQ-ACDC integration, as shown in the Figure 5.

The specific algorithm is as follows:

F Artifacts and Reproducibility

F.1 Software Dependencies and Licenses

Our implementation builds upon several open-source software libraries, all of which are licensed for academic and commercial use:

Deep Learning Frameworks:

- **PyTorch** (BSD-3-Clause License): We use PyTorch as the primary deep learning framework for model implementation and tensor operations.

- **Hugging Face Transformers** (Apache 2.0 License): Used for loading pre-trained lan-

Algorithm 1 PAHQ Integration with ACDC (Accelerated Mixed-Precision Strategy)

Require: Model parameters θ, clean input X, corrupted input $\tilde{X}$, pruning thresholds τ, δ.
Require: Weights available in FP32 (W_{32}) and pre-loaded FP8 (W_8).
Ensure: Sparse circuit G_{sparse} and task metric.

Initialization: $G = (V, E)$, $E_{prev} \leftarrow \emptyset$, importance scores $I(e) \leftarrow 0$.
while not converged ($|E|$ stabilizes or steps $< T$) **do**
  Update baseline activations: $A_{clean} \leftarrow \text{Infer}(X)$; $A_{corrupt} \leftarrow \text{Infer}(\tilde{X})$
  for each edge $e = (u \rightarrow v)$ in E **do**
    // Mixed-Precision Activation Patching
    Let l_s, h_s be the layer and head of source node u.
    for layer $l = 1$ to L **do**
      Identify critical head $h^* \leftarrow (l == l_s ? h_s : \emptyset)$.
      Parallel Execution Strategy:
        • *Stream 1 (Background):* Compute FP8 Attention A_8 for all heads using W_8.
        • *Stream 2 (Priority):* If h^* exists, async load $W_{32}^{(h^*)}$ and compute FP32 Attention $A_{32}^{(h^*)}$.
      Synchronization: Wait for streams. Merge $A \leftarrow A_8$, overwrite $A^{(h^*)} \leftarrow A_{32}^{(h^*)}$ if active.
      Compute Output: $Y^{(l)} \leftarrow \text{Proj}(A) \cdot W_O$ (Mixed Precision).
      if $l == l_s$ **then**
        $Y^{(l,h_s)} \leftarrow A_{clean}^{(l_s,h_s)}$ **// Patching Operation**
      end if
      $\tilde{X}^{(l+1)} \leftarrow \text{FFN}(\text{LayerNorm}(Y^{(l)}))$
    end for
    // Importance Evaluation
    Let $A_{patched}$ be the final activations at destination v.
    Metric Change $\Delta = ||A_{corrupt}^{(v)} - A_{patched}||$.
    $I(e) \leftarrow (\Delta > \delta) ? \Delta : 0$.
  end for
  // Pruning Step
  $E \leftarrow \{e \in E \mid I(e) > \tau\}$. Update G.
end while
return G

guage models and tokenizers. All models are accessed through the Hugging Face Model Hub[4].

Mechanistic Interpretability Tools:

- **TransformerLens** (Nanda and Bloom, 2022) (MIT License): A library specifically designed for mechanistic interpretability research, providing utilities for analyzing transformer activations and circuit discovery.

All dependencies use permissive open-source licenses that permit modification and redistribution, ensuring full reproducibility of our work.

[4] https://huggingface.co/models

F.2 Models and Datasets

F.2.1 Pre-trained Models

We evaluate PAHQ on three publicly available transformer models:

- **GPT-2** (Radford et al., 2019): A 124M parameter autoregressive language model trained on WebText. Available under a modified MIT license through Hugging Face (`gpt2`).
- **Attn-4l** (Heimersheim and Janiak, 2023): A 4-layer attention-only transformer designed for interpretability research. Publicly available under MIT license.
- **Redwood-2l** (Gokaslan and Cohen, 2019): A 2-layer transformer model trained on OpenWebText. Available under Apache 2.0 license.

All models were downloaded from Hugging Face Model Hub using their official model identifiers. We did not modify model weights or architectures; all experiments use the original pre-trained checkpoints.

F.2.2 Evaluation Tasks and Datasets

We evaluate on three standard mechanistic interpretability benchmarks:

- **Indirect Object Identification (IOI)** (Wang et al., 2022): Tests the model's ability to identify indirect objects in sentences. The dataset consists of 1,000 synthetic examples with templates like "When [Name1] and [Name2] went to the store, [Name2] gave a gift to [Name1]." We converted the original token-based format to natural language and re-tokenized using each model's vocabulary.
- **Docstring** (Heimersheim and Janiak, 2023): Evaluates Python function documentation generation from code. The dataset contains 10,000 function-docstring pairs from open-source Python repositories. We use the publicly released version from the ACDC repository.
- **Greater Than** (Nanda et al., 2023): A numerical reasoning task requiring comparison of two numbers. The dataset includes 2,000 number pairs with varying magnitudes and digit lengths.

All datasets are either synthetically generated or derived from publicly available code repositories, containing no personally identifiable information.

F.3 Data Privacy and Ethical Considerations

Pre-training Data: The pre-trained models we use (GPT-2, Attn-4l, Redwood-2l) were trained on large-scale internet text corpora. While we did not participate in the original data collection or model training, we acknowledge:

- These corpora may contain personal information that was publicly available on the internet at the time of collection
- The training data may include content that could be considered offensive or biased
- We use these models solely for analyzing their internal mechanisms and do not generate or analyze content targeting individuals or groups

Evaluation Data: Our evaluation datasets are either:

- Synthetically generated (IOI, Greater Than) with no real personal information
- Derived from open-source code repositories (Docstring) with public licenses

No additional data collection or human subject involvement occurred in this research.

F.4 Intended Use and Compatibility

PAHQ is designed specifically for mechanistic interpretability research and maintains compatibility with the original intended uses of all utilized artifacts:

- **PyTorch and Transformers**: Used within their designed scope for deep learning research and inference
- **TransformerLens**: Applied precisely for its intended purpose of mechanistic interpretability analysis
- **Pre-trained models**: Used only for analyzing internal mechanisms, not for downstream generation tasks
- **Circuit discovery methods**: PAHQ accelerates existing methods (ACDC) without modifying their algorithmic logic or intended research applications

Our method introduces quantization as an inference optimization technique while preserving the faithfulness and interpretability guarantees of the underlying circuit discovery algorithms. All experiments operate within the research contexts intended by the original artifact creators.

F.5 Code and Implementation Details

Our implementation will be made publicly available upon publication to facilitate reproduction and extension of this work. The codebase includes:

- Custom CUDA kernels for mixed-precision attention computation
- Three-stream parallel scheduler implementation
- Integration with ACDC and other circuit discovery methods

- Evaluation scripts for all three benchmark tasks
- Detailed documentation and usage examples

Hardware Requirements: All experiments were conducted on NVIDIA H20-NVLink GPUs (96GB) with 16 vCPU AMD EPYC 9K84 processors. The method is compatible with any CUDA-capable GPU supporting FP8 operations (compute capability >=8.9).

Software Requirements:

- Python >= 3.8
- PyTorch >= 2.0 with CUDA support
- Transformers >= 4.30
- TransformerLens >= 1.0

Complete installation instructions and dependency specifications will be provided in the code repository.

G Artifacts and Reproducibility

G.1 Software Dependencies and Licenses

Our implementation builds upon several open-source software libraries, all of which are licensed for academic and commercial use:

Deep Learning Frameworks:

- **PyTorch** (BSD-3-Clause License): We use PyTorch as the primary deep learning framework for model implementation and tensor operations.
- **Hugging Face Transformers** (Apache 2.0 License): Used for loading pre-trained language models and tokenizers. All models are accessed through the Hugging Face Model Hub[5].

Mechanistic Interpretability Tools:

- **TransformerLens** (Nanda and Bloom, 2022) (MIT License): A library specifically designed for mechanistic interpretability research, providing utilities for analyzing transformer activations and circuit discovery.

All dependencies use permissive open-source licenses that permit modification and redistribution, ensuring full reproducibility of our work.

[5] https://huggingface.co/models

G.2 Models and Datasets

G.2.1 Pre-trained Models

We evaluate PAHQ on three publicly available transformer models:

- **GPT-2** (Radford et al., 2019): A 124M parameter autoregressive language model trained on WebText. Available under a modified MIT license through Hugging Face (`gpt2`).
- **Attn-4l** (Heimersheim and Janiak, 2023): A 4-layer attention-only transformer designed for interpretability research. Publicly available under MIT license.
- **Redwood-2l** (Gokaslan and Cohen, 2019): A 2-layer transformer model trained on OpenWebText. Available under Apache 2.0 license.

All models were downloaded from Hugging Face Model Hub using their official model identifiers. We did not modify model weights or architectures; all experiments use the original pre-trained checkpoints.

G.2.2 Evaluation Tasks and Datasets

We evaluate on three standard mechanistic interpretability benchmarks:

- **Indirect Object Identification (IOI)** (Wang et al., 2022): Tests the model's ability to identify indirect objects in sentences. The dataset consists of 1,000 synthetic examples with templates like "When [Name1] and [Name2] went to the store, [Name2] gave a gift to [Name1]." We converted the original token-based format to natural language and re-tokenized using each model's vocabulary.
- **Docstring** (Heimersheim and Janiak, 2023): Evaluates Python function documentation generation from code. The dataset contains 10,000 function-docstring pairs from open-source Python repositories. We use the publicly released version from the ACDC repository.
- **Greater Than** (Nanda et al., 2023): A numerical reasoning task requiring comparison of two numbers. The dataset includes 2,000 number pairs with varying magnitudes and digit lengths.

All datasets are either synthetically generated or derived from publicly available code repositories, containing no personally identifiable information.

G.3 Data Privacy and Ethical Considerations

Pre-training Data: The pre-trained models we use (GPT-2, Attn-4l, Redwood-2l) were trained on large-scale internet text corpora. While we did not participate in the original data collection or model training, we acknowledge:

- These corpora may contain personal information that was publicly available on the internet at the time of collection
- The training data may include content that could be considered offensive or biased
- We use these models solely for analyzing their internal mechanisms and do not generate or analyze content targeting individuals or groups

Evaluation Data: Our evaluation datasets are either:

- Synthetically generated (IOI, Greater Than) with no real personal information
- Derived from open-source code repositories (Docstring) with public licenses

No additional data collection or human subject involvement occurred in this research.

G.4 Intended Use and Compatibility

PAHQ is designed specifically for mechanistic interpretability research and maintains compatibility with the original intended uses of all utilized artifacts:

- **PyTorch and Transformers**: Used within their designed scope for deep learning research and inference
- **TransformerLens**: Applied precisely for its intended purpose of mechanistic interpretability analysis
- **Pre-trained models**: Used only for analyzing internal mechanisms, not for downstream generation tasks
- **Circuit discovery methods**: PAHQ accelerates existing methods (ACDC) without modifying their algorithmic logic or intended research applications

Our method introduces quantization as an inference optimization technique while preserving the faithfulness and interpretability guarantees of the underlying circuit discovery algorithms. All experiments operate within the research contexts intended by the original artifact creators.

G.5 Code and Implementation Details

Our implementation will be made publicly available upon publication to facilitate reproduction and extension of this work. The codebase includes:

- Custom CUDA kernels for mixed-precision attention computation
- Three-stream parallel scheduler implementation
- Integration with ACDC and other circuit discovery methods
- Evaluation scripts for all three benchmark tasks
- Detailed documentation and usage examples

Hardware Requirements: All experiments were conducted on NVIDIA H20-NVLink GPUs (96GB) with 16 vCPU AMD EPYC 9K84 processors. The method is compatible with any CUDA-capable GPU supporting FP8 operations (compute capability >=8.9).

Software Requirements:

- Python >= 3.8
- PyTorch >= 2.0 with CUDA support
- Transformers >= 4.30
- TransformerLens >= 1.0

Complete installation instructions and dependency specifications will be provided in the code repository.