Boolean-aware Attention for Dense Retrieval

Anonymous ACL submission

Abstract

We present Boolean-aware attention, a novel attention mechanism that dynamically adjusts token focus based on Boolean operators (e.g., "and", "not"). Our model employs specialized Boolean experts, each tailored to amplify or suppress attention for operator-specific contexts. A predefined gating mechanism activates the corresponding experts based on the detected Boolean type. Experiments on two Boolean retrieval datasets demonstrate that integrating *BoolAttn* with BERT greatly enhances the model's capability to process Boolean queries.

1 Introduction

Boolean operations such as *and*, *or*, and *not* are fundamental components of human logic and reasoning, serving as essential tools for combining, including, or excluding specific information in queries. These operations are particularly valuable when users have complex and precise information needs, such as "Movies set in Vietnam but *not* about war" or "Animals found only in Brazil *and* Mexico." However, traditional lexical matching methods like BM25 (Robertson et al., 2009) often fail to adequately handle queries involving exclusions, frequently retrieving results that include undesired information. Similarly, transformer-based architectures such as BERT (Devlin, 2018) encounter challenges in processing logical operations (Mai et al., 2024; Malaviya et al., 2023). These models lack explicit mechanisms to interpret Boolean cues and struggle with contextual dependencies, leading to suboptimal performance in handling logical semantics. To overcome these limitations, we present a Boolean-aware attention mechanism, an extension of the standard attention mechanism, designed to explicitly model Boolean logic with enhanced sensitivity to logical cues and their contextual scopes. Our method ensures that models can better capture and process logical constructs, enabling improved performance on tasks involving complex Boolean queries.

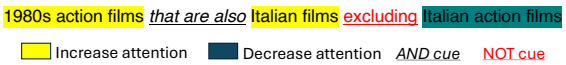

Figure 1: Token-wise attention scores are adjusted based on Boolean type. Tokens affected by *and* will receive more attention while tokens affected by *not* will receive less attention score.

2 Boolean-aware attention

Our Boolean-aware attention mechanism (or, *BoolAttn*, for short) consists of four key components: (1) a **Cue Predictor** (2.1) that detects Boolean cues within the input sequence, (2) a **Contextual Scope Predictor** (2.2) that determines the scope of Boolean operators, ensuring their influence is properly localized, (3) a **Bias Predictor** (2.3) that adjusts attention scores based on Boolean operations using operator-specific embeddings and positional dependencies, and (4) a **modified attention mechanism** incorporating Boolean-specific biases into the attention scores. By explicitly modeling Boolean logic while preserving contextual dependencies, *BoolAttn* enables transformers to better interpret logical structures in natural language queries. Furthermore, *BoolAttn* is lightweight, modular, and fully compatible with existing transformer architectures, serving as a *plug-and-play* enhancement for self-attention layers. The overall architecture of *BoolAttn* is illustrated in Figure 2.

Boolean operator embeddings We use operator-specific embeddings $\mathbf{o} \in \mathbf{R}^{d_o}$ to encode the unique semantic roles of Boolean operators, such as the negation effect of *not* or the conjunctive nature of *and*. These embeddings guide other modules to produce operator-specific outputs, enabling the model to adapt to the logical properties of each operator.

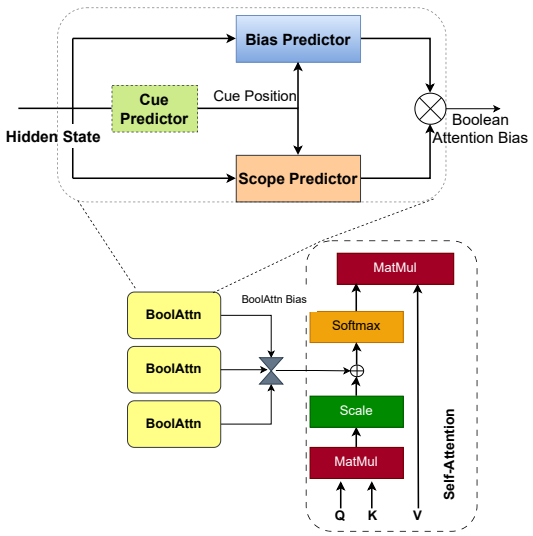

Figure 2: Boolean-aware Attention architecture.

2.1 Boolean Cue Predictor

Boolean cues play a pivotal role in determining the logical structure of queries. These cues dictate the scope and interaction between different query components, thereby influencing the embedding of the query. For instance, cues for *not* (negation) include keywords like "not," "excluding," and "without," while cues for *and* include "and," "as well as," "also," and "including." The Cue Predictor (CuePred) is designed to identify the positions of Boolean cues within the input sequence. Since the positional information of cues remains invariant for a given input sequence, the CuePred is employed exclusively in the first transformer layer. The detected cue positions are then propagated through subsequent layers to inform the Scope Predictor and Bias Predictor at each corresponding layer. Notably, Boolean-related retrieval datasets (Malaviya et al., 2023) do not explicitly annotate Boolean cues. Therefore, cue detection is formulated as an auxiliary task during the pretraining phase, enabling the model to learn cue positions implicitly and generalize effectively to downstream tasks.

The operator embeddings $\mathbf{o}$ are projected into the model's hidden space using a linear transformation

$$\mathbf{o}' = \mathbf{W_o}\mathbf{o} + \mathbf{b}_o$$

where $\mathbf{W}_o \in \mathbf{R}^{d_h \times d_o}$ and $\mathbf{b}_o \in \mathbf{R}^{d_h}$ are learnable parameters, d_h and d_o denote the hidden and operator embedding dimensions, respectively. The projected embedding is then added to the hidden states H before going through a shared linear layer to compute cue scores, followed by sigmoid activation function to compute the cue probabilities:

$$p_c = \sigma(\mathbf{W}_c(\mathbf{H} + \mathbf{o}') + \mathbf{b_c})$$

where $\mathbf{W}_c \in \mathbf{R}^{1 \times d_h}$ and $\mathbf{b}_c \in \mathbf{R}$ are learnable parameters. Cue probabilities $p_c \in \mathbf{R}^{B \times L}$ indicates the likelihood of each position being a cue.

2.2 Boolean Scope Predictor

The Scope Predictor (ScopePred) identifies the affected span of tokens (the *scope*) for a given Boolean operator. For example, in the sentence shown in Figure 1, the scope of *not* is "Italian action film," and the model downweights the importance of these tokens in the *BoolAttn* mechanism. The predictor leverages local contextual features and conditions on operator embeddings to determine the scope. It combines a Conv1D layer to capture local dependencies and a FiLM layer (Feature-wise Linear Modulation, Perez et al., 2018) to specialize outputs for each operator.

Boolean operators primarily influence tokens near the cue, as observed in phrases "not happy," where negation affects the adjacent token. To effectively capture such local dependencies, ScopePred utilizes a Conv1D layer. Especially, the hidden states $H \in \mathbf{R}^{B \times L \times d_h}$ and cue probabilities $p_c \in \mathbf{R}^{B \times L}$ are concatenated and subsequently passed through a Conv1D layer:

$$S = \text{Conv1D}([\mathbf{H}; \mathbf{p_c}])$$

This layer efficiently captures local patterns and dependencies around the cue, making it particularly suitable for modeling the impact of Boolean operators on nearby tokens. To adapt the predictions for each Boolean operator, the FiLM layer modulates the Conv1D output using operator embeddings $\mathbf{o}$. Scale and shift factors are computed as:

$$\gamma = \mathbf{W}_\gamma o + \mathbf{b}_\gamma, \quad \beta = \mathbf{W}_\beta o + \mathbf{b}_\beta$$

These factors are then applied to modulate the Conv1D output:

$$S = \gamma S + \beta$$

The modulated logits S are passed through a sigmoid activation to obtain scope probabilities:

$$P_{scope} = \sigma(S)$$

Since the scope is binary (0 for outside scope, 1 for inside scope), we use the Gumbel-Sigmoid trick to

enable discrete scope sampling while preserving differentiability. The scope is determined by applying a learnable threshold θ to the probabilities P obtained from the Gumbel-Sigmoid:

$$\mathrm{S} = \mathbf{I}(P > \theta) \tag{1}$$

To allow gradient flow through the non-differentiable thresholding, we use a Straight-Through Estimator (STE). This approach ensures that the model learns to make discrete scope predictions effectively.

Different Boolean operators impose unique structural constraints on token interactions, necessitating specialized mechanisms for scope prediction. The *and* operator establishes bidirectional reinforcement between operands (or entities), ensuring mutual influence, whereas *not* applies a unidirectional effect, modifying the meaning of specific tokens within its scope. When a negation cue like "not" appears, it alters the interpretation of a single token or phrase rather than forming a bidirectional relationship. Conversely, *and* enforces a reciprocal dependency—e.g., in “A and B," both A and B should influence each other—requiring a pairwise rather than token-wise scope. To capture these differences, we define the scope matrices for *and* and *or* as follows:

$$S_{mutual} = S \cdot S^T$$

where $S_{mutual} \in \mathbf{R}^{B \times heads \times L \times L}$ captures symmetric token influence, while *not* retains a unary mask of shape $(B, heads, L, 1)$ as it requires no pairwise expansion.

2.3 Bias Predictor

We use a Boolean-aware bias module to model token-level biases in *BoolAttn* by incorporating positional dependencies and operator-specific embeddings. This module learns to adjust attention bias based on the relative positions of Boolean cue (from the CuePred) and the surrounding context.

Relative position To capture the structural influence of Boolean operators, the model first computes the relative positions of tokens with respect to the Boolean cues (e.g., “without"). Given a sequence of length L, the relative position of token i with respect to a Boolean cue at position c is defined as:

$$r_i = i - c$$

For the negation operator (*not*), its influence is asymmetric, affecting only subsequent tokens. To ensure this, we clip negative relative postions:

$$r_i = \max(0, i - c), \text{ for } \textit{not}$$

For conjunctions (*and*) and disjuntions (*or*), influence is bidirectional, meaning the relative positions are clamped with a symmetric window $\pm d$:

$$r_i = \text{clip}(i - c, -d, d), \text{ for } \textit{and /or}$$

To assign different levels of importance based on distance from the cue, we can use position embeddings. To make this module lightweight, instead of using learnable positional embeddings, we use a Gaussian kernel to weight positions. The idea of using Gaussian kernel is we hypothesize that tokens closer to the cue are more likely to be influenced:

$$w(r_i) = \exp\left(-\frac{r_i^2}{2 * \sigma^2}\right)$$

where σ is a learnable parameter controlling the spread of the Gaussian influence. The computed positional weights $w(r)$ are concatenated with token hidden state H to form an enriched representation before passed through a Feed-Forward Network (FFN) to generate context-aware bias scores:

$$\mathbf{b} = \text{FFN}([\mathbf{H}, w(r)])$$

To account for the distinct biases associated with different Boolean operations, an operator-specific gate dynamically modulates the computed biases:

$$\mathbf{b}' = \mathbf{b} \cdot \sigma(\mathbf{W}_{op}\mathbf{e}_{op}) \tag{2}$$

The final bias values are regularized using softplus to explicitly ensure non-negativity.

2.4 Operator-gating mechanism

After having Scope (Eq. 1) and Bias (Eq. 2) of each Boolean operator, we integrate the Boolean reasoning into the transformer’s attention mechanism. This integration employs a gating mechanism, dynamically weighting each operator’s influence based on its presence in the input.

The attention bias contribution for a Boolean operator *op* is formulated as:

$$S_{op} = G_{op} \cdot \text{Scope}_{op} \cdot \text{Bias}_{op}$$

where $G_{op} \in \{0; 1\}$ is a gating variable that indicates the presence of the operator in the input sequence. This gate can be either learned through an

experts classifier or provided as an auxiliary label, enabling conditional application of the attention bias based on the occurrence of the operator. Since we aim to reinforce tokens of *and*, *or* and down-weight the importance of tokens affected by *not* (Appendix A), the final Boolean-aware attention bias is then formulated as:

$$S_{Boolean} = S_{and} + S_{or} - S_{not}$$

The Boolean-aware bias is added to the raw attention scores, ensuring that the logical structure of sentences influences the attention mechanism:

$$\text{Attention Score} = \text{Softmax}(\frac{\mathbf{QK}^T}{\sqrt{d}} + S_{Boolean})\mathbf{V}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are original attention matrices in the self-attention mechanism (Vaswani, 2017).

3 Evaluation

We evaluate *BoolAttn* by integrating it into BERT (*uncased* configuration), forming *Bool-BERT*, and testing it on two Boolean retrieval datasets: Quest (Malaviya et al., 2023) and BoolQuestions (Zhang et al., 2024). Prior to the main evaluation, we pretrain *BoolAttn* on the GPT-generated dataset by Mai et al., 2024 to ensure effective initialization (see Appendix B). All experiments are repeated three times with different random seeds {0, 42, 1234}, and we report the average results.

Entity-seeking queries QUEST features queries with implicit Boolean operations, testing models' ability to understand logical relationships among entities. We employ Bool-BERT as the query encoder in a dual-encoder setup, paired with BERT as the document encoder. Retrieval performance is measured using average Recall@K, with K set to $\{20, 50, 100, 1000\}$, as shown in Table 1. Additionally, we assess the retrievers on seven distinct query structures to capture nuances in retrieval capabilities, with results presented in Figure 3. These findings demonstrate that integrating the *BoolAttn* plugin enhances BERTs' effectiveness in Boolean query understanding.

BoolQuestions We use the best-performing models from fine-tuning on Quest as starting checkpoints for continuous learning on BoolQuestions' supplementary training dataset, focusing on *and*, *or*, and *not* operations. Evaluation is conducted on the corresponding test set, adhering to the original experimental setup by reporting Mean Reciprocal Rank at top-10 (MRR@10). The results in Table 2 highlight the effectiveness of *BoolAttn* in enhancing performance for Boolean-related QA tasks. For comparison, the results of *distillbert-dot-v5* are taken from the original paper.

	Avg. Recall@K			
Retriever	20	50	100	1000
BERT-*base*	0.228	0.344	0.443	0.760
Bool-BERT-*base*	0.254	0.379	0.483	0.788
BERT-*large*	0.238	0.364	0.467	0.792
Bool-BERT-*large*	0.266	0.395	0.495	0.812

Table 1: Bool-BERT retrievers outperform standard BERT-based models in retrieval tasks.

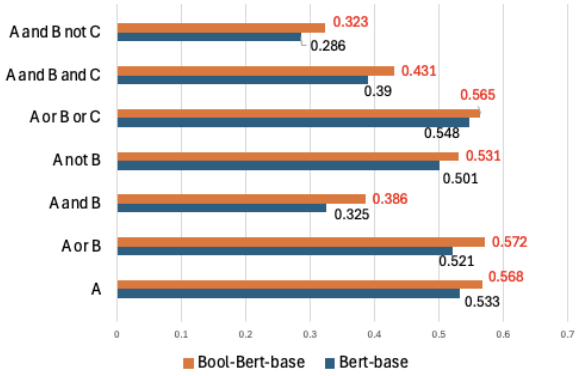

Figure 3: Recall@100 across different query templates.

4 Conclusion

In this work, we introduced *BoolAttn*, a novel attention mechanism that dynamically adjusts attention weights based on Boolean operators, effectively capturing the distinct structural influences of conjunctions, disjunctions, and negations. By integrating *BoolAttn* into BERT, we greatly improved performance on Boolean retrieval tasks, demonstrating its effectiveness in processing complex logical relationships within queries. These results highlight the potential of Boolean-aware attention mechanisms to enhance transformers' ability to understand and handle nuanced Boolean queries.

	MRR@10		
	AND	OR	NOT
distillbert-dot-v5	0.399	0.530	0.130
BERT-*base*	0.412	0.545	0.128
Bool-BERT-*base*	0.480	0.586	0.165
Bool-BERT-*large*	0.532	0.610	0.182

Table 2: Performance on MS MARCO's BoolQuestions.

Limitations

While *BoolAttn* greatly enhances the performance of the backbone BERT model and is designed to be lightweight, it still introduces a considerable number of parameters. Specifically, Bool-BERT-*base* has approximately 121M parameters, while the *large* version reaches around 378M parameters. This indicates that *BoolAttn* adds roughly 10% more parameters to the underlying transformer architecture, increasing *computation cost and latency*. To mitigate this overhead, future work could explore parameter-sharing strategies across Boolean modules or investigate low-rank factorization techniques to reduce the parameter count without sacrificing performance.

Moreover, the Scope and Bias Predictors depend on accurate inputs from the Cue Predictor, making the system vulnerable to error propagation. If cues are misidentified, scope and bias predictions may be incorrect. This is particularly challenging when Boolean cues are implied rather than explicit. For example, in the query "Find articles on climate policy favoring renewables over fossil fuels," the comparative phrase implicitly negates fossil fuels without an explicit *not* operator. If the Cue Predictor misses this, the system might retrieve documents about both "renewables" and "fossil fuels", misinterpreting the intended negation. This could lead to misleading search results, especially in *sensitive domains* like legal, medical, or financial information retrieval.

Additionally, the absence of explicit ground truth for scope labels in existing Boolean retrieval datasets presents a significant challenge. Since no direct supervision is available for scope boundaries, the optimization of *BoolAttn* relies on indirect signals from retrieval performance, which may not fully capture the intended scope dynamics. This lack of supervision makes it difficult to ensure that *BoolAttn* learns the desired scope representation, potentially leading to suboptimal generalization.

Lastly, the limited size of Boolean retrieval datasets constrains the scalability of *BoolAttn*. Although the model is effective in enhancing Boolean query understanding, the small training corpus limits its potential to learn complex linguistic patterns and generalize to diverse scenarios. This *scalability issue* poses challenges for deploying *BoolAttn* in large-scale retrieval systems, as its performance may degrade without sufficient training data.

Ethical considerations

This work utilizes Quest, BoolQuestions, and GPT-generated datasets strictly for research purposes, consistent with their original licensing terms. The *BoolAttn* module is designed for academic evaluation of Boolean retrieval models. Any derivative models or results are intended solely for research and should not be used outside of academic contexts. We ensured that the datasets used in this study, including QUEST and BoolQuestions, did not contain personally identifiable information (PII) or offensive content. A thorough review was conducted to confirm that the data complied with ethical standards.

References

Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Carolin Dudschig, Barbara Kaup, Mingya Liu, and Juliane Schwab. 2021. The processing of negation and polarity: An overview. *Journal of Psycholinguistic Research*, 50(6):1199–1213.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

Quan Mai, Susan Gauch, and Douglas Adams. 2024. Setbert: Enhancing retrieval performance for boolean logic and set operation queries. *arXiv preprint arXiv:2406.17282*.

Chaitanya Malaviya, Peter Shaw, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2023. Quest: A retrieval dataset of entity-seeking queries with implicit set operations. *arXiv preprint arXiv:2305.11694*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In

Proceedings of the AAAI conference on artificial intelligence, volume 32.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Daniel M Wegner. 1994. Ironic processes of mental control. *Psychological review*, 101(1):34.

Thomas Wolf. 2020. Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Zongmeng Zhang, Jinhua Zhu, Wengang Zhou, Xiang Qi, Peng Zhang, and Houqiang Li. 2024. Boolquestions: Does dense retrieval understand boolean logic in language? *arXiv preprint arXiv:2411.12235*.

A Why downweight attention scores for negation

Negation treatment is a challenging problem for current dense retrieval models (Zhang et al., 2024; Malaviya et al., 2023; Mai et al., 2024).

The phenomenon where attempting to suppress specific thoughts leads to their increased prominence is well-documented in psychological literature. This is exemplified by the "pink elephant" paradox: when instructed not to think of a pink elephant, ones often find it challenging to avoid the image. This counterproductive effect is known as ironic process theory (Wegner, 1994), which posits that deliberate efforts to suppress certain thoughts can make them more persistent.

In the context of language processing, negation requires the brain to first represent the concept being negated before it can suppress or modify it. This two-step process suggests that negation involves initial activation followed by inhibition (Dudschig et al., 2021).

Building upon these insights, our hypothesis proposes that in Boolean-aware attention mechanisms, tokens affected by negation should be downweighted. By reducing the influence of these negated tokens, the model can more accurately capture the intended logical structure of the input, aligning with the cognitive processes involved in handling negation.

B Pretrain Boolean-aware Attention

Introducing new parameters in *BoolAttn* poses a risk of overfitting due to the limited size of Boolean retrieval datasets compared to the model's capacity when integrated into transformer layers. This imbalance may lead the model to memorize the training data rather than generalizing effectively, resulting in suboptimal performance. To mitigate this, we pretrain Bool-BERT on Mai et al., 2024's GPT-generated dataset on two auxiliary tasks: cue prediction and expert classification (gate selection). The cue position prediction loss is calculated as follows:

$$\mathcal{L}_{cue} = -\frac{1}{B \times L} \sum_{i=1}^{B} \sum_{j=1}^{L} (y_{i,j} \log p_{i,j} + (1 - y_{i,j}) \log(1 - p_{i,j})$$

where y is the ground-truth label and p is the predicted probability that token j in batch i is a cue. The original dataset only includes "and," "or," and "not" as Boolean cues. To enhance model robustness, we augment the dataset by replacing cues with their semantic equivalents, such as substituting "and" with "as well as," "are also," or "including," and replacing "not" with "other than" or "excluding." Consequently, multiple cue tokens can appear in a single sequence. We use BERT's tokenizer, followed by a word-matching approach to label the cue tokens.

The second task in pretraining phase is *Booealn expert classfication*. This task is a multi-label classification problem designed to identify which Boolean operators are present in a given input sequence. Since a sequence may contain more than one Boolean operation, this task requires predicting multiple labels simultaneously (one for each operator). Given E Boolean experts (3 if we use *and*, *or*, *not*), the loss is defined as

$$\mathcal{L}_{gate} = -\frac{1}{B \times E} \sum_{i=1}^{B} \sum_{k=1}^{E} (y_{i,k} \log p_{i,k} + (1 - y_{i,k}) \log(1 - p_{i,k})$$

where y is the ground-truth label and $p_{i,k}$ is the predicted probability for operator k in batch i.

Until now we only pretrain the CuePred and Gate modules, the parameters for ScopePred and BiasPred remain uninitialized. To provide these modules with a reasonable starting point without overfitting them to Boolean logic, we train them on the Boolean sentence classification task from Mai et al., 2024 using a standard triplet loss. Although Mai et al., 2024 reported that triplet loss hinders

performance on Boolean queries, we deliberately use it to provide an "okay" initialization, allowing the modules to learn actively during the main tasks. The pretraining objective is defined as follows:

$$\mathcal{L}_{pretrain} = \mathcal{L}_{cue} + \mathcal{L}_{gate} + \alpha\mathcal{L}_{triplet}$$

where α is set to a small value, we used 0.2. During pretraining, only the *BoolAttn* modules and BERT's embedding layer are updated, while the other layers are frozen. We train for a single epoch with a low learning rate of $1e-5$.

C BoolAttn hyperparameters and settings

All experiments are conducted on two NVIDIA Quadro RTX 8000 GPUs, each with 50GB of memory. We use PyTorch (Paszke et al., 2019) as our deep learning framework and leverage Hugging Face's Transformers library (Wolf, 2020) for pretrained models and model implementation.

Hyperparameters (1) Operator embedding dimension d_h: 10. (2) Threshold θ is initialized at 0.5. (3) Relative postion window: 5. (4) Gaussian kernal sigma: σ: 2.

Gating mechanism For the learnable gate mechanism, we employ a simple FFN followed by a sigmoid activation, with logits scaled by a small temperature value of 0.1 to enhance sensitivity. Unlike softmax, which enforces mutual exclusivity, sigmoid is better suited for Boolean logic since multiple Boolean operators can coexist within a single query. By applying a hard threshold to the sigmoid outputs, the gates can selectively activate multiple Boolean experts, allowing them to contribute to the final attention scores as needed.

The gating mechanism introduces a performance bottleneck as it relies on accurately activating the Boolean experts corresponding to the operators present in the query. This dependency can lead to suboptimal performance (Table 3). Improving this gating strategy is left for future work. However, in real-world applications, users could explicitly specify the Boolean logic in their queries. This information could then be used as an auxiliary label to activate the relevant Boolean experts—an approach we currently employ.

D Dense Retriever

We adopt a dual-encoder framework (Karpukhin et al., 2020) for our retriever.

	Avg. Recall@K		
Retriever	50	100	1000
Bool-Bert-*base*	0.379	0.483	0.788
Bool-Bert-*base*-gate	0.371	0.470	0.775

Table 3: Learnable gate introduces bottleneck.

	Avg. Recall@K		
Retriever	50	100	1000
BERT-Base	0.134	0.182	0.453
BERT-Large	0.227	0.300	0.627
T5-Base	0.372	0.455	0.726
T5-Large	**0.386**	**0.476**	0.757
BERT-Base-Ours	0.344	0.443	**0.760**

Table 4: Importance of choosing different positive samples, results for QUEST dataset.

Unlike Malaviya et al., 2023 and Mai et al., 2024, which use a fixed positive document per query, we sample different positive documents at each training step. This encourages the encoders to explore diverse relevant documents for the same query, which is crucial for multi-answer retrieval tasks where multiple correct answers exist. This is particularly important for boolean queries: negation excludes a subset of answers, disjunctions (unions) expand the answer set, and conjunctions (intersections) narrow it. By introducing varied positive samples, our approach enhances model robustness and generalization. Furthermore, we empirically found that using random negative samples is more effective than hard negative samples (i.e., incorrect answers retrieved by BM25), particularly for the QUEST dataset. Table 4 shows the improvement when using different positive documents and normal negative samples, T5 and BERT results taken from Malaviya et al., 2023 and Mai et al., 2024, respectively.

Hyperparameters

1. Max query length: 64.
2. Max document length: 256.
3. Negative document per sample: 5.
4. Positive document: 1, shuffled every step.
5. Batch size: 32.
6. Optimizer: AdamW (Kingma and Ba, 2015) with weight decay 0.01 (Loshchilov and Hutter, 2019).

7. Learning rate: 5e-5.
8. Epoch: 40.
9. Evaluation every epoch, keep best.