EyeLayer: Integrating Human Attention Patterns into LLM-Based Code Summarization

Anonymous Author(s)

Abstract

Code summarization is the task of generating natural language descriptions of source code, which is critical for software comprehension and maintenance. While large language models (LLMs) have achieved remarkable progress on this task, an open question remains: can human expertise in code understanding further guide and enhance these models? We propose EyeLayer, a lightweight attention-augmentation module that incorporates human eye-gaze patterns, as a proxy of human expertise, into LLM-based code summarization. EyeLayer models human attention during code reading via a Multimodal Gaussian Mixture, redistributing token embeddings based on learned parameters (μ_i, σ_i^2) that capture where and how intensively developers focus. This design enables learning generalizable attention priors from eye-tracking data and incorporating them into LLMs seamlessly, without disturbing existing representations. We evaluate EyeLayer across diverse model families (i.e., LLaMA-3.2, Qwen3, and CodeBERT) covering different scales and architectures. EyeLayer consistently outperforms strong fine-tuning baselines across standard metrics, achieving gains of up to 13.17% on BLEU-4. These results demonstrate that human gaze patterns encode complementary attention signals that enhance the semantic focus of LLMs and transfer effectively across diverse models for code summarization.

Keywords

Code Summarization, Human Factors in Software Engineering, Human-centered AI for Software Engineering

ACM Reference Format:
Anonymous Author(s). 2018. EyeLayer: Integrating Human Attention Patterns into LLM-Based Code Summarization. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Software documentation is an essential bridge between code implementation and developer understanding, with code summarization facilitating efficient program comprehension [1, 46]. As modern software systems become increasingly complex, quickly grasping code functionality through concise summaries is critical for maintenance and evolution. Consequently, automatically generating high-quality summaries has become a central challenge in software engineering [24, 37].

Recent advances in large language models (LLMs) have demonstrated remarkable capabilities in code-related tasks, particularly in code summarization [11, 21, 46]. While these models have achieved good performance by learning from vast corpora of code–summary pairs, there remains a gap in generating human-aligned summaries that capture the information humans actually focus on during code comprehension[2, 5]. Meanwhile, when developers comprehend code to formulate summaries, their attention patterns reveal how they selectively allocate focus across different parts of the code [24, 40]. In previous software engineering research, eye-tracking studies have been widely used to extract developers' attention patterns which is a promising proxy for their cognitions during programming activities [39, 41, 42]. This motivates a key question: **can incorporating human attention signals further enhance LLM-based code summarization?**

The most recent research has attempted to guide AI model development leveraging developers' attention patterns and demonstrated promising benefits of such guidance. EyeTrans [55] for the first time integrated eye-gaze signals into a single Transformer block for code summarization, achieving up to 6.39% improvement. However, it remains unknown whether human attention can actually enhance modern LLMs, which differ substantially in scale, architecture, and optimization dynamics. This uncertainty limits their potential impact on real-world applications.

To bridge the gap between human and LLM attention mechanisms, we propose **EyeLayer**, a lightweight architectural module that integrates human eye-gaze data into LLM-based code summarization. Our approach is grounded in a key insight: during code comprehension, programmers naturally focus their attention unevenly across the code, concentrating intensively on semantically critical regions while peripherally attending to contextual elements. EyeLayer models this distributional attention as a transferable prior, learned from a curated eye-tracking corpus of 27 professional developers [55], which captures how human gaze behavior reflects semantic importance during real code comprehension. It employs a **Multimodal Gaussian Mixture** to redistribute each code embedding based on learned parameters (μ_i, σ_i^2), which encode both the intensity and spread of human attention. Integrated into the supervised fine-tuning process, EyeLayer leverages these human-derived priors to improve how pretrained models allocate focus across code tokens without altering the original model architecture. Despite being trained on a small but cognitively grounded dataset, EyeLayer generalizes effectively to large-scale LLMs, showing that even limited human attention data can yield measurable improvements.

Functionally, EyeLayer serves as a recommendation system for code embedding redistribution: for each code embedding, it predicts a small set of Gaussian modes that recommend how its representation should be redistributed. This mechanism allows the model to compose fine-grained and global focus patterns, analogous to personalized recommendation in representation space. By decoupling gaze-informed redistribution from the model's intrinsic attention weights, EyeLayer learns generalizable attention priors from sparse eye-tracking data and transfers them to unseen code. Incorporated within LLMs, it preserves pretrained representations while infusing human-like focus behavior directly into the attention redistribution process.

We evaluate EyeLayer across five models spanning different scales and architectures: CodeBERT (125M) [14], LLaMA-3.2-1B/3B-Instruct [18], and Qwen3-1.7B/4B-base [48]. All EyeLayer-augmented models are compared against strong supervised fine-tuned baselines trained on identical code summarization data (CodeXGLUE [22, 29]) but without eye-tracking integration, isolating the contribution of human attention signals. Evaluation uses four widely-adopted metrics capturing lexical overlap (BLEU[35], ROUGE-L[27], METEOR[4]) and semantic similarity (BERTScore[52]). Across all five models, EyeLayer achieves consistent gains over fine-tuning baselines, with improvements up to 13.17% on BLEU-4, confirming that human attention signals enhance LLM performance across architectures.

This paper makes the following contributions:

- We propose a framework for integrating human cognitive priors into large language models for code summarization. Using eye-tracking data as transferable probabilistic priors, our approach establishes a bridge between human attention behavior and LLM-level attention formation.
- We design the *Multimodal Gaussian EyeLayer*, a lightweight, recommendation-like module that redistributes code embeddings through learnable Gaussian mixtures. This mechanism decouples gaze-informed redistribution from intrinsic attention weights, enabling scalable integration of sparse human signals into billion-parameter LLMs without disrupting pretrained representations.
- We conduct a systematic evaluation across five LLMs spanning both encoder-only and decoder-only architectures, demonstrating consistent improvements on the CodeXGLUE benchmark and strong transferability of learned attention priors to unseen code.
- To facilitate reproducibility and foster future research, we release our implementation scripts and datasets at **URL**.

In the rest of this paper, Section 2 presents the background of eye-tracking in program comprehension and probabilistic attention modeling. Section 3 introduces the design and implementation details of the proposed *EyeLayer* architecture. Section 4 details the experimental setup. Section 5 analyze the results. Section 6 discusses potential threats to validity. Section 7 provides a broader discussion of findings and implications. Section 8 reviews related work. Finally, Section 9 concludes the paper and outlines directions for future research.

2 Background

Human gaze behavior offers empirical insight into how developers comprehend code, while probabilistic attention provides a principled way to model such focus computationally. This section reviews key findings from eye-tracking studies and links them to Gaussian-based attention formulations that inspire our EyeLayer design.

2.1 Eye-tracking for Program Comprehension

Eye-tracking has become a rigorous method for examining cognitive processes in software engineering research, particularly in understanding how developers read and comprehend source code [17, 40]. By capturing gaze behavior, eye-tracking enables the quantitative analysis of attention allocation and processing effort with high temporal precision. In software engineering, this relationship is particularly relevant because program comprehension, like natural language reading, involves the incremental interpretation of complex visual and semantic structures [41]. Fixation-based metrics provide a means to infer where and when developers engage in information processing, distinguishing meaningful cognitive activity from mere visual transitions represented by saccades [39].

The theoretical basis for interpreting gaze data originates from cognitive psychology, most notably the work of Just and Carpenter [23]. Their eye–mind assumption states that the duration of a fixation, the period of relative ocular stability directly reflects the time required for cognitive processing. This principle established fixations as a reliable indicator of comprehension effort in reading, linking visual attention to linguistic and semantic processing. Empirical evidence shows that fixations occupy the vast majority of viewing time during code reading, emphasizing their role as the fundamental unit of analysis for understanding comprehension behavior [40, 41]. Overall, fixation analysis offers a direct and interpretable connection between observable gaze patterns and the underlying cognitive mechanisms of program understanding, making eye-tracking a valuable empirical approach for investigating how developers read, reason about, and make decisions based on source code.

2.2 Probabilistic Attention and Cognitive Priors

Transformer attention can be framed probabilistically, with weights parameterized as continuous distributions over positions. Gaussian parameterizations offer a simple and interpretable form: a mean for focus location and a variance for spread. Representative studies show concrete uses of such priors. Chorowski et al. introduced Gaussian-shaped attention for sequence-to-sequence alignment in speech recognition [8]. Cordonnier et al. analyzed self-attention and showed that learned patterns relate closely to Gaussian-like kernels over relative positions [10]. You et al. further reported that hard-coded Gaussian windows can match the performance of fully learned attention in machine translation, indicating that Gaussian structure can serve as an effective bias [49]. To allow multiple foci, Graves modeled attention as a mixture of Gaussians in recurrent architectures, capturing multi-modal alignments with learnable centers and spreads [19].

This probabilistic view aligns with findings from eye-tracking. Studies in software engineering report localized and selective fixations during code reading [7, 42]. Such fixation maps are commonly summarized as peaked distributions over spatial locations. Neural models inspired by selective vision, such as DRAW, use parameterized Gaussian filters to realize differentiable focus regions [20]. These results motivate representing model attention with Gaussian or mixture forms when human-like focus is desirable.

Guided by this evidence, our EyeLayer treats attention as a learnable mixture with sparse mode selection. The formulation provides an interpretable parameter space (centers, spreads, and weights) consistent with probabilistic attention and with observed fixation patterns in code comprehension. This connects a statistical prior on attention with cognitively grounded signals in a single mechanism.

3 Methodology

Our training employs a carefully designed dual-dataset strategy that separates the primary code summarization task from the auxiliary eye-tracking alignment task, enabling learning from both large-scale code-summary pairs and sparse but cognitively grounded attention signals. Figure 1 provides an overview of our complete approach.

3.1 Datasets and Preprocessing

Our training employs a carefully designed dual-dataset strategy that separates the primary code summarization task from the auxiliary eye-tracking alignment task, enabling learning from both large-scale code-summary pairs and sparse but cognitively grounded attention signals.

Code Summarization Corpus. We use the Java subset of the CodeXGLUE benchmark [29], a widely-adopted dataset containing Java methods paired with their corresponding docstring summaries extracted from open-source repositories. The dataset provides diverse code patterns spanning different programming idioms, complexity levels, and documentation styles, enabling robust learning of the code-to-summary mapping across varied contexts.

Eye-Tracking Corpus. We derive the auxiliary alignment supervision from the EyeTrans corpus [55], which records the gaze behaviors of developers during controlled code comprehension tasks. Each sample links a Java method to its Abstract Syntax Tree (AST) and corresponding fixations that capture how programmers allocate attention across syntactic and semantic regions. A *fixation* is defined as a spatially stable gaze lasting approximately 100–300 ms [40], during which most visual information processing occurs [23]. Each fixation is localized on screen coordinates and mapped to its corresponding AST node, producing discrete yet cognitively grounded attention signals. These node-level fixation counts are then aligned to model-level subtoken representations through our three-stage matching pipeline described below, providing precise human-derived supervision for multimodal alignment in the EyeLayer.

To effectively integrate these fixation-based signals into the model, we must reconcile the representational gap between the human gaze space and the model input space. The eye-tracking corpus encodes attention in the *AST node space*, identifying which syntactic constructs programmers focus on, whereas the multimodal EyeLayer operates in the *subtoken space*, defined by byte-pair encoded tokens from the model tokenizer. This mismatch is non-trivial: (1) a single AST node like `BFSdistance` may split into multiple subtokens [`BFS`, `distance`], (2) tokenization varies based on surrounding context and instruction templates, and (3) abstract AST nodes have no direct token correspondence. We address this through a three-stage alignment pipeline: first, we traverse the AST to extract concrete code elements; second, we apply context-aware tokenization matching the model's instruction format; finally, we use multi-strategy matching—exact matching for simple cases, consecutive aggregation for split tokens, and character offset estimation for complex constructs—to map AST nodes to subtoken indices. This pipeline achieves >98% mapping accuracy and enables us to transfer sparse node-level fixation counts to the dense subtoken representations required for attention supervision.

Independent Data Sources. To ensure clear supervision boundaries, the two datasets are kept entirely independent. The code summarization corpus drives the primary generation objective, while the eye-tracking corpus contributes auxiliary alignment supervision. They contain disjoint code samples, eliminating data leakage and ensuring that observed improvements stem from the integration of human cognitive priors rather than exposure to additional labeled summaries.

3.2 Multimodal Gaussian EyeLayer

Our approach builds on the key insight that, during code comprehension, programmers allocate attention unevenly across the code: they concentrate intensively on semantically critical regions while attending peripherally to contextual elements. This uneven distribution can be viewed as a composition of several focus patterns, each representing a localized concentration of attention over the token sequence. To model this behavior, the **Multimodal Gaussian EyeLayer** represents attention as a mixture of Gaussian components. Each component defines a focus region characterized by a center μ_k (semantic locus) and spread σ_k (contextual extent), while a sparse gating network determines how many such regions are needed for each code snippet. This formulation captures both concentrated and distributed focus within a unified probabilistic framework, allowing the model to adaptively modulate attention according to code structure and semantics.

The EyeLayer integrates into pretrained decoder-only transformers (e.g., LLaMA, Qwen) through hook-based injection at an intermediate layer. During forward propagation, it intercepts hidden states $\mathbf{H}$, applies the EyeLayer transformation, and returns updated representations $\mathbf{H}'$ to subsequent layers. This hook-based design preserves the causal structure of the base model while enriching its intermediate representations with human-aligned attention priors, as illustrated in Figure 2.

3.2.1 *Code-Level Embedding.*

Before predicting Gaussian parameters, the model first summarizes the overall semantic context of the input sequence. For hidden states $\mathbf{H} \in \mathbb{R}^{B \times L \times d}$ from an intermediate transformer layer, we apply an attention mask $\mathbf{M}_{\text{attn}}$ and a special-token mask $\mathbf{M}_{\text{special}}$ to form $\mathbf{M} = \mathbf{M}_{\text{attn}} \odot (1 - \mathbf{M}_{\text{special}})$. When positional information is available, a decay factor $D_{p_i} = \gamma^{p_i}$ ($\gamma = 0.95$) down-weights distant tokens. The code-level embedding is computed as:

$$\mathbf{e} = \frac{\sum_{i=1}^{L} M_i D_{p_i} \mathbf{H}_i}{\sum_{i=1}^{L} M_i + \epsilon}, \tag{1}$$

where ϵ is a small constant to avoid division by zero. The resulting vector $\mathbf{e} \in \mathbb{R}^d$ provides a compact semantic summary for mode prediction.

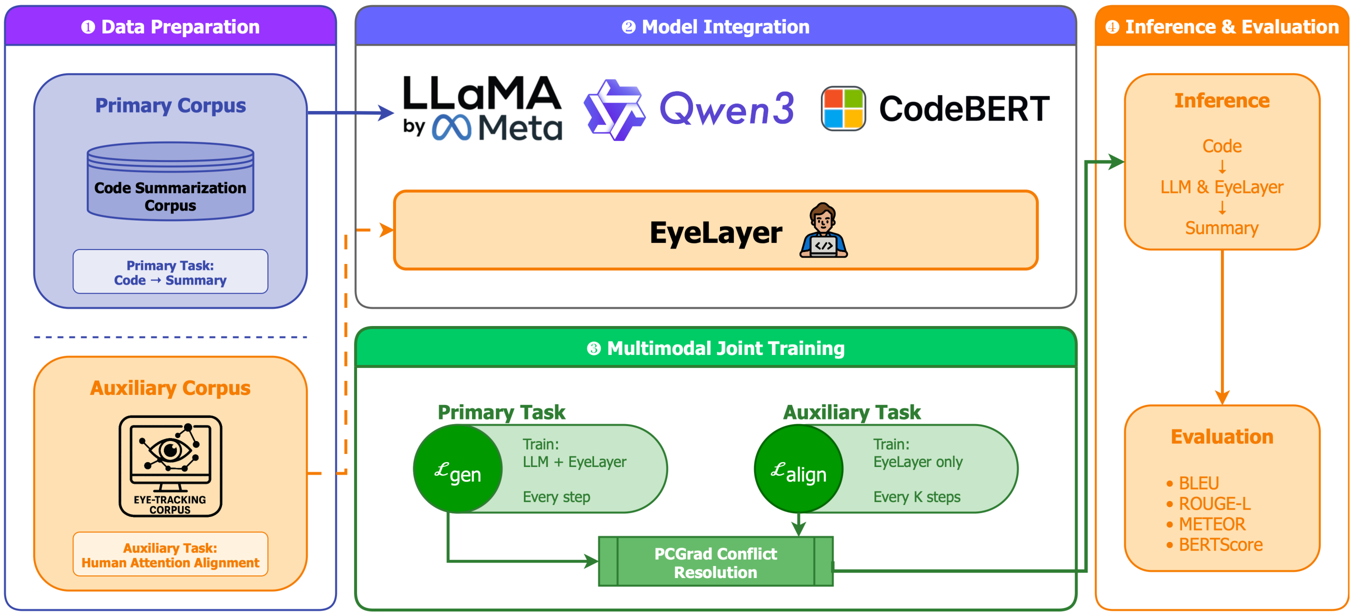


Figure 1: Overview of our joint training pipeline.

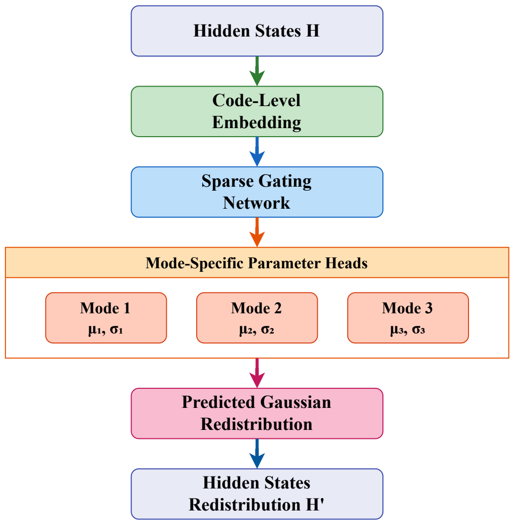


Figure 2: The Multimodal Gaussian EyeLayer architecture.

3.2.2 *Sparse Gating Mechanism.* To decide how many Gaussian components should be activated for each code sequence, the EyeLayer uses a lightweight gating network that maps the code embedding $\mathbf{e}$ to a normalized weight vector $\mathbf{w} \in \mathbb{R}^K$:

$$\mathbf{w} = \text{softmax}(\mathbf{W}_2\, \phi(\mathbf{W}_1 \mathbf{e} + \mathbf{b}_1) + \mathbf{b}_2)\,, \tag{2}$$

where $\phi(\cdot)$ is a non-linear activation, and $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$ are learnable projection and bias parameters. The softmax normalization ensures $\sum_k w^{(k)} = 1$, yielding interpretable mode activations that indicate the relative contribution of each Gaussian component. This gating mechanism encourages sparse activation: simple functions tend to concentrate weight on a single mode, whereas more complex code distributes attention across multiple regions. Such adaptive allocation allows the model to adjust its focus continuously without introducing discrete decisions or additional supervision.

3.2.3 *Mode-Specific Parameterization.* Each active mode predicts its Gaussian parameters based on shared semantic features extracted from the same code embedding $\mathbf{e}$. The shared representation is computed as:

$$\mathbf{h}_{\text{shared}} = \text{Dropout}(\text{LayerNorm}(\text{GELU}(\mathbf{W}_h \mathbf{e} + \mathbf{b}_h))), \tag{3}$$

where $\mathbf{W}_h$ and $\mathbf{b}_h$ are learnable projection parameters. Each mode then applies lightweight linear heads:

$$\tilde{\mu}_k = \mathbf{w}_\mu^{(k)} \mathbf{h}_{\text{shared}} + b_\mu^{(k)}, \tag{4}$$

$$\tilde{\sigma}_k = \mathbf{w}_\sigma^{(k)} \mathbf{h}_{\text{shared}} + b_\sigma^{(k)}, \tag{5}$$

where $\tilde{\mu}_k$ and $\tilde{\sigma}_k$ are raw predictions for the center and spread of the k-th Gaussian component. Predictions are constrained to $\mu_k \in [0, L-1]$ and $\sigma_k \in [\sigma_{\min}, L/2]$ to ensure valid ranges. Centroid biases are initialized to cover early, middle, and late regions of the sequence to promote spatial diversity during early training.

3.2.4 *Gaussian Mixture Construction.* The final attention distribution is formed as a weighted mixture of K=3 Gaussian components:

$$P(i) = \sum_{k=1}^{K} w^{(k)} \frac{\exp\left(-\frac{(i-\mu_k)^2}{2\sigma_k^2}\right)}{\sum_{j=1}^{L} \exp\left(-\frac{(j-\mu_k)^2}{2\sigma_k^2}\right)}, \tag{6}$$

where $P(i)$ denotes the predicted attention probability for token position i in a sequence of length L. Each token position corresponds to a code token aligned with an AST node, thus representing a specific syntactic or semantic unit in the source code. $w^{(k)}$ is the normalized weight of the k-th mode, and the denominator ensures each Gaussian is properly normalized over all token positions. Smaller σ_k values produce sharper, concentrated peaks representing focused reading, whereas larger σ_k values yield broader distributions that capture peripheral attention. The resulting mixture $P(i)$ forms a smooth, interpretable, and differentiable attention distribution that aligns with empirical human fixation patterns and supports end-to-end optimization. The resulting distribution (P(i)) serves as the human-aligned attention prior used in the subsequent causal-aware redistribution stage (Section 3.3).

3.3 Causal-Aware Attention Redistribution

Integrating human-guided attention into decoder-only transformers predicted by the EyeLayer requires preserving their causal autoregressive dependency. Unlike encoder-based models that permit bidirectional attention, decoder-only architectures must maintain strict left-to-right information flow so that each token prediction depends only on preceding context. Directly modifying attention weights or masks would break this constraint and disrupt key–value caching during generation. To address this, we implement *causal-aware redistribution*, which injects human-aligned guidance through residual perturbations of hidden states rather than altering attention masks. The perturbation is shaped by the Gaussian attention distribution predicted by the EyeLayer, enabling soft alignment toward human-attended regions while fully preserving causality. The mechanism proceeds in three stages: (1) low-rank transformation for compact perturbation generation, (2) attention-guided weighting for cognitively informed modulation, and (3) adaptive gating for dynamic integration control.

3.3.1 *Low-Rank Transformation.*

To prevent overfitting on limited eye-tracking data, perturbations are generated through a low-rank bottleneck. Given hidden states $\mathbf{H} \in \mathbb{R}^{B \times L \times d}$ from a target transformer layer, we first down-project and then reconstruct the representations:

$$\mathbf{Z} = \mathrm{ReLU}(\mathbf{H}\mathbf{W}_{\mathrm{down}}), \tag{7}$$

$$\Delta\mathbf{H}_{\mathrm{base}} = \mathbf{Z}\mathbf{W}_{\mathrm{up}}, \tag{8}$$

where $\mathbf{W}_{\mathrm{down}} \in \mathbb{R}^{d \times r}$ and $\mathbf{W}_{\mathrm{up}} \in \mathbb{R}^{r \times d}$ are learnable projections with rank $r \ll d$. This factorization requires only $2dr$ parameters instead of d^2, providing a 64× reduction when $r = 16$ for $d = 2048$, while retaining sufficient representational capacity.

3.3.2 *Attention-Guided Weighting.*

The perturbation is reweighted according to the predicted Gaussian attention distribution, emphasizing regions that align with human gaze. For each sample b and token position i, we compute:

$$\tilde{\Delta}\mathbf{H}_{b,i} = \lambda \, P_b(i) \, \Delta\mathbf{H}_{\mathrm{base},b,i} \odot A_i, \tag{9}$$

where $P_b(i)$ denotes the mixture-based attention probability at position i, λ is a learnable scaling coefficient, $A_i \in \{0, 1\}$ marks valid token positions, and $\odot$ represents element-wise multiplication. Since redistribution operates on hidden representations rather than attention masks, causal self-attention remains intact: each token still attends only to past positions ($j \le i$), while its representation is softly modulated toward human-attended regions. Gradient clipping is applied to ensure numerical stability.

3.3.3 *Adaptive Gating and Integration.*

Finally, an adaptive highway gate controls the strength of human-guided perturbation for each sample. A scalar gate value $g_b \in [0, g_{\max}]$ is computed as:

$$g_b = g_{\max} \, \sigma(\mathrm{MLP}([\bar{\mathbf{h}}_b; \mathbf{f}_b])), \tag{10}$$

where $\bar{\mathbf{h}}_b = \frac{1}{L}\sum_{i=1}^{L} \mathbf{H}_{b,i}$ is the mean-pooled hidden state (layer-normalized before concatenation), $\mathbf{f}_b$ encodes global statistics of the attention distribution (e.g., entropy, maximum probability, and in the multimodal case, mode count and weight entropy), and $\sigma(\cdot)$ is the sigmoid activation. The MLP is initialized with a negative bias to encourage conservative gating during early training. The final hidden states are obtained via residual integration:

$$\mathbf{H}'_{b,i} = \mathbf{H}_{b,i} + \alpha \, g_b \, \tilde{\Delta}\mathbf{H}_{b,i}, \tag{11}$$

where α is a global scaling constant. When g_b is small, the EyeLayer exerts minimal influence; as g_b increases, stronger redistribution occurs, enabling adaptive incorporation of human attention signals while preserving the model's pretrained representations.

3.4 Model Integration

The Multimodal EyeLayer integrates with transformer architectures through strategies that respect their information flow, as shown in Figure 3.

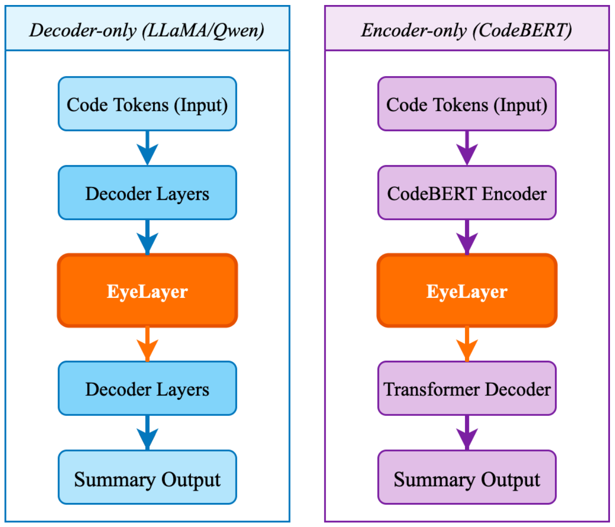


Figure 3: Integration of the EyeLayer into transformer architectures for code summarization. Note that since CodeBERT is an encoder-only model, an auxiliary decoder is attached for sequence generation in the code summarization task.

Decoder-Only Models (LLaMA, Qwen). For autoregressive decoder-only architectures, the EyeLayer is injected at an intermediate transformer layer. During forward propagation, when the base model reaches the target layer, the hook intercepts hidden states $\mathbf{H}$, applies the EyeLayer transformation, and returns enhanced representations $\mathbf{H}'$ to subsequent layers. The predicted distribution $P(i)$

guides causal-aware attention redistribution (Section 3.3), which enforces that token i only attends to positions $j \le i$ and preserves the decoder's generation order.

Encoder-Only Models (CodeBERT). For encoder-only architectures, the EyeLayer operates after CodeBERT and before the auxiliary decoder. CodeBERT processes the input code to produce contextualized hidden states $\mathbf{H}_{\text{enc}}$, which are pooled to obtain a global code embedding that drives gating and mode prediction. The resulting $P(i)$ modulates $\mathbf{H}_{\text{enc}}$ via a non-causal low-rank perturbation over token positions; causal masking is not applied because the encoder is bidirectional. The decoder then cross-attends to the modulated encoder representations enriched with human-aligned attention priors.

3.5 Joint Training

After integrating the EyeLayer into the model architecture, we jointly train the system on the primary code summarization and auxiliary eye-tracking alignment tasks. This joint learning setup allows the model to balance large-scale textual supervision with sparse but cognitively grounded human signals. Formally, the overall objective combines a generation loss $\mathcal{L}_{\text{gen}}$ and an auxiliary alignment loss $\mathcal{L}_{\text{align}}$ (defined in Section 4.3.2):

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{gen}} + \lambda_{\text{align}}\, \mathcal{L}_{\text{align}}, \tag{12}$$

where λ_{align} is a small weighting coefficient that ensures the alignment supervision acts as a regularizer rather than dominating optimization.

3.5.1 Interleaved Training Schedule. Because the two datasets differ greatly in scale, with tens of thousands of code-summary pairs and only hundreds of eye-tracking samples, we adopt an interleaved training schedule to maintain stability. During each epoch, the model primarily trains on the summarization dataset, updating parameters with $\mathcal{L}_{\text{gen}}$ at every step. Every K steps (typically $K = 200$), a batch from the eye-tracking dataset is inserted, and EyeLayer is optimized jointly on $\mathcal{L}_{\text{total}}$ with gradient conflict handling described in Section 3.5.2. At the end of each epoch, we conduct a dedicated alignment sweep over the entire eye-tracking dataset while freezing the base model parameters, updating only the EyeLayer components. This two-phase schedule maintains consistent exposure to the generation objective and provides sufficient gradient signal for the EyeLayer through dedicated alignment phases, preventing the alignment objective from being overshadowed by the main summarization task.

3.5.2 Projecting Conflicting Gradients (PCGrad). Multi-task optimization often leads to conflicting gradient directions between objectives. In our setting, the EyeLayer parameters are influenced by both $\mathcal{L}_{\text{gen}}$ and $\mathcal{L}_{\text{align}}$, which may occasionally compete. To reconcile these objectives, we employ **Projecting Conflicting Gradients (PCGrad)** [50], which detects negative cosine similarity between task gradients and removes the conflicting component through orthogonal projection. When gradients are aligned, both signals are preserved; when they diverge, PCGrad adjusts each gradient to retain only the non-conflicting directions. The final parameter update uses the mean of the projected gradients, ensuring that human-guided supervision complements rather than disrupts the main learning objective.

4 Experimental Setup

This section details the experimental configuration used to evaluate the proposed Multimodal Gaussian EyeLayer. We describe (1) dataset construction for both code summarization and eye-tracking supervision, (2) models and training infrastructure, and (3) evaluation metrics for summarization quality and human attention alignment. These components collectively establish the framework for answering the research questions presented in Section 5.

4.1 Datasets

Code Summarization Dataset. We use a subset of CodeXGLUE [29], derived from CodeSearchNet-Java, as the primary supervision source. To reduce training cost while preserving data diversity, we sample 10% of the corpus, yielding 16,492 training pairs, 518 validation pairs, and 1,095 test pairs. Each instance consists of a Java method paired with its corresponding docstring summary extracted from open-source repositories.

Eye-Tracking Dataset. We adopt the EyeTrans corpus [55] for human attention supervision. The corpus involves fixation data from 27 programmers performing code summarization tasks. Each data point corresponds to a unique (developer, method) pair, covering 64 unique functions across diverse Java projects. We obtain 625 annotated samples with fixation sequences aligned to AST nodes. These samples provide sparse but cognitively grounded supervision for guiding attention redistribution.

4.2 Models and Training Infrastructure

We evaluate our Multimodal Gaussian EyeLayer across three representative transformer architectures spanning different scales and designs: LLaMA3.2-1B and LLaMA3.2-3B (decoder-only instruction-tuned models), Qwen3-1.7B and Qwen3-4B (decoder-only base model), and CodeBERT (encoder-only code model). Training and evaluation are conducted on a single NVIDIA L40S GPU (45GB VRAM), confirming that our approach remains computationally efficient while effectively incorporating human attention guidance.

4.3 Evaluation Metrics

4.3.1 Code Summarization Metrics. We evaluate generation quality using four widely adopted metrics:

- **BLEU** [35]: Computes modified n-gram precision with a brevity penalty to quantify lexical overlap with references.
- **ROUGE-L** [27]: Measures F1 based on the longest common subsequence, reflecting sequence-level similarity.
- **METEOR** [4]: Aligns words using exact, stem, and synonym matches with fragmentation penalties, emphasizing recall and paraphrase recognition.
- **BERTScore** [52]: Computes contextual embedding similarity to assess semantic alignment between candidate and reference texts.

4.3.2 Attention Alignment Metrics. To align model-predicted attention with human fixation patterns while preserving multimodal diversity, we define:

$$\mathcal{L}_{\text{align}} = \mathcal{L}_{\text{match}} + \lambda_{\text{sep}} \mathcal{L}_{\text{MSP}}, \tag{13}$$

where $\mathcal{L}_{\text{match}}$ aligns each Gaussian mode with fixation data, and $\mathcal{L}_{\text{MSP}}$ enforces spatial separation among active modes. The matching term is computed as $\mathcal{L}_{\text{match}} = \sum_{k=1}^{K} \tilde{w}^{(k)} \sum_t \lambda_t \mathcal{L}_t^{(k)}$, where $t \in \{\text{CAL}, \text{SML}, \text{CR}, \text{AUP}\}$ and $\tilde{w}^{(k)}$ is the normalized mode weight. Here $w^{(k)}$, μ_k, and σ_k denote the weight, center, and spread of the k-th Gaussian; $P_k(i)$ is its normalized probability at token position i; $F(i)$ is the human fixation frequency; and ϵ is a small stability constant.

Centroid Alignment (CAL). $\mathcal{L}_{\text{CAL}}^{(k)} = \sqrt{(\mu_k - \mu_{\text{human}})^2 + \epsilon}$, where μ_{human} is the empirical fixation centroid. This term aligns each predicted attention center μ_k with human focus regions.

Spread Matching (SML). $\mathcal{L}_{\text{SML}}^{(k)} = \sqrt{(\sigma_k - \sigma_{\text{target}})^2 + \epsilon}$, where σ_{target} represents the observed fixation spread. It ensures each mode captures realistic human attention breadth.

Concentration Reward (CR). $\mathcal{L}_{\text{CR}}^{(k)} = 1 - \left(\sum_{i \in \mathcal{W}} P_k(i)\right)^2$, where $\mathcal{W}$ is a local window around attended tokens. This rewards probability mass concentrated near human fixation areas.

Anti-Uniform Penalty (AUP). $\mathcal{L}_{\text{AUP}}^{(k)} = \max(0, c - D_{\text{KL}}(U \| P_k))$, where $U(i) = 1/L$ is the uniform baseline, D_{KL} is KL divergence, and c is a small positive margin controlling the penalty strength. This term penalizes near-uniform distributions and promotes sharper attention focus.

Mode Separation (MSP). $\mathcal{L}_{\text{MSP}} = \sum_{k_1 < k_2} s_{k_1 k_2} \max(0, m - |\mu_{k_1} - \mu_{k_2}|)$, where $s_{k_1 k_2} = \mathbb{I}[w^{(k_1)} > \tau]\mathbb{I}[w^{(k_2)} > \tau]$, τ is the activation threshold, and m is the minimum distance between active mode centers. This term maintains spatial diversity across Gaussian components.

5 Experimental Results and Analysis

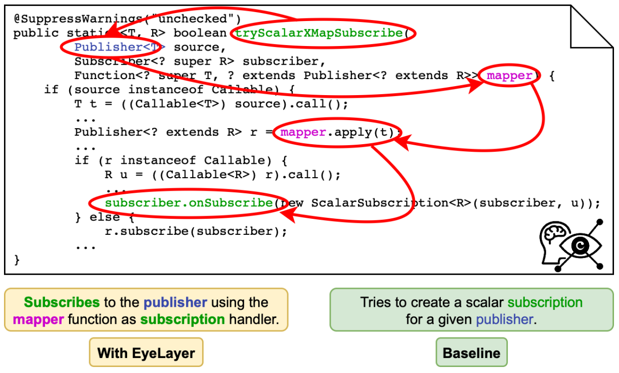
To evaluate the proposed Multimodal Gaussian EyeLayer, we address four research questions designed to quantify its effect on model performance, architectural behavior, and design components.

- **RQ1 – Does EyeLayer improve code summarization quality compared to standard supervised finetuning?**
- **RQ2 – How does the position of the EyeLayer within the transformer stack influence performance?**
- **RQ3 – How effectively does the EyeLayer generalize to encoder-only architectures?**
- **RQ4 – How does EyeLayer multimodal design contribute to performance?**

5.1 RQ1: Effectiveness Compared to SFT

RQ1 investigates whether integrating the proposed EyeLayer improves code summarization quality compared to standard supervised finetuning (SFT) without eye-tracking guidance. We evaluate four representative models: instruction-tuned (Llama3.2-1B/3B) and base (Qwen3-1.7B/4B).

As shown in Table 1, integrating EyeLayer leads to consistent gains across all models and evaluation metrics. Improvements appear in both lexical metrics (BLEU, ROUGE, METEOR) and semantic similarity (BERTScore), suggesting that cognitively inspired attention cues can guide the model toward more functionally meaningful code regions. For instruction-tuned models (Llama3.2-instruct), EyeLayer yields steady gains, particularly for the 1B model (+1.8 BLEU-4 / +1.9 METEOR). For base models(Qwen3), the improvement is



```
@SuppressWarnings("unchecked")
public static <T, R> boolean tryScalarXMapSubscribe(
        Publisher<T> source,
        Subscriber<? super R> subscriber,
        Function<? super T, ? extends Publisher<? extends R>> mapper) {
    if (source instanceof Callable) {
        T t = ((Callable<T>) source).call();
        ...
        Publisher<? extends R> r = mapper.apply(t);
        ...
        if (r instanceof Callable) {
            R u = ((Callable<R>) r).call();
            ...
            subscriber.onSubscribe(new ScalarSubscription<R>(subscriber, u));
        } else {
            r.subscribe(subscriber);
        ...
}
```

Subscribes to the publisher using the mapper function as subscription handler.
With EyeLayer
Tries to create a scalar subscription for a given publisher.
Baseline

Figure 4: Example from CodeXGLUE illustrating EyeLayer's improvement over the baseline. Depict the inferred gaze-inspired attention across semantically related code regions.

larger in absolute terms, particularly for Qwen3-1.7B (ROUGE-L: +5.28, METEOR: +5.43), which indicates that models lacking instruction-level supervision may benefit more from additional attention prior.

The performance improvement suggests that EyeLayer subtly guides intermediate attention toward critical code regions that typically attract human gaze, thereby improving the quality of generated summary. The relatively larger gains observed in smaller models imply that supervision from the eye-tracking corpus provides a more informative inductive signal when model capacity and learned abstractions are limited. Larger models which already develop rich internal attention patterns, exhibit smaller yet consistent benefits. These observations collectively point to EyeLayer as a light but effective cognitive guidance mechanism, offering additional structure to models operating under supervision.

Figure 4 illustrates a representative example that demonstrates how EyeLayer enhances the generated summary. The baseline output, *"Tries to create a scalar subscription for a given publisher,"* captures only surface lexical cues, whereas EyeLayer produces a more accurate behavioral description, *"Subscribes to the publisher using the mapper function as subscription handler."* Compared to the baseline, EyeLayer places stronger focus on the method declaration and variable declarations, which are semantically critical regions for capturing functional intent. This pattern resonates with the human attention dynamics reported by Karas et al. [24], where programmers most frequently alternate their gaze between method declarations and variable declarations during code comprehension. The correspondence suggests that EyeLayer internalizes similar focus tendencies without explicit gaze supervision during inference, enabling the model to generalize cognitive attention patterns that guide summarization toward semantically informative code regions.

RQ1 Summary. EyeLayer consistently improves summarization across all models, with larger gains in smaller or less supervised settings, showing that lightweight cognitive cues enhance semantic focus in code comprehension.

Table 1: Performance comparison of baseline models and models with EyeLayer integration. Values in parentheses denote absolute improvement over the SFT baseline.

Model	BLEU-4	ROUGE-L	METEOR	BERTScore
Llama3.2-1B	14.31	22.12	27.45	87.55
Llama3.2-1B + EyeLayer	16.18 (**+1.87**)	23.51 (**+1.39**)	29.33 (**+1.88**)	88.51 (**+0.96**)
Llama3.2-3B	15.64	24.57	29.83	88.29
Llama3.2-3B + EyeLayer	16.86 (**+1.22**)	25.25 (**+0.68**)	31.04 (**+1.21**)	88.72 (**+0.43**)
Qwen3-1.7B	13.36	21.39	26.60	86.04
Qwen3-1.7B + EyeLayer	15.12 (**+1.76**)	26.67 (**+5.28**)	32.03 (**+5.43**)	86.38 (**+0.34**)
Qwen3-4B	15.24	23.73	29.45	85.87
Qwen3-4B + EyeLayer	17.22 (**+1.98**)	25.30 (**+1.57**)	31.31 (**+1.86**)	86.27 (**+0.40**)

5.2 RQ2: Effect of EyeLayer Insertion Position

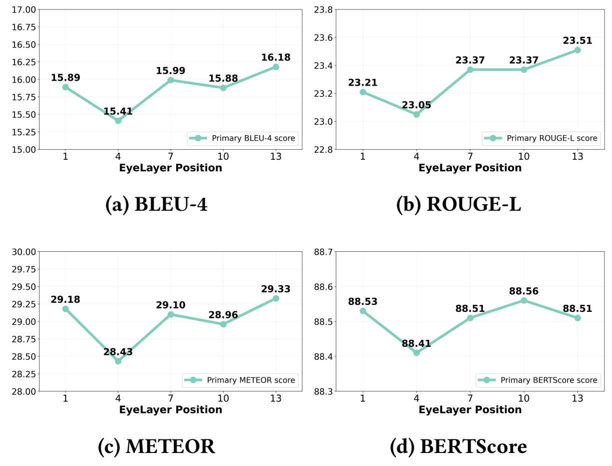


Figure 5: Performance of Llama3.2-1B-Instruct when the EyeLayer is inserted at different transformer layers.

We investigate how integration depth affects performance by inserting the EyeLayer into different transformer layers of Llama3.2-1B-Instruct (16 layers). Figure 5 shows the different metric trends across positions.

Two clear patterns emerge: (1) performance improves toward deeper layers and peaks at layer13, and (2) a temporary drop appears around layer 4. This trend aligns with the hierarchical roles of transformer layers [32, 44]. Early layers capture lexical and syntactic features, middle layers integrate contextual semantics, and later-middle layers refine coherent representations for generation [13]. The degradation at layer 4 likely reflects interference with unstable intermediate encodings, as this stage is still reorganizing shallow features into higher-level structures [53]. At layer 13, semantic representations are largely formed yet remain adaptable. Injecting human attention priors here allows modulation of semantic focus without disrupting earlier composition, enhancing alignment with meaningful program structures [32, 44].

Overall, these results highlight that the integration of cognitive priors depends strongly on the model's representational stage, with later-middle layers providing the best balance between semantic completeness and flexibility [13, 53].

RQ2 Summary. Performance peaks at later-middle layers, where semantic representations are mature yet flexible, indicating that cognitive priors are most effective after semantic integration but before generation.

5.3 RQ3: Generalization to Encoder-Only Architectures

Building on the results from decoder-only models (RQ1) and the optimal integration depth analysis (RQ2), RQ3 examines whether EyeLayer generalizes to encoder-only architectures, which differ fundamentally in information flow and attention dynamics. We evaluate this transferability using CodeBERT with the encoder-side integration strategy described in Section 3.4. The results are summarized in Table 2.

EyeLayer maintains consistent improvements across all metrics, despite the architectural shift from decoder-only to encoder-only models. The largest gain appears in METEOR (+1.83), indicating enhanced semantic alignment and paraphrase understanding, both of which rely on holistic code comprehension. The bidirectional encoder benefits from modeling human-like focus over the entire code context without causal masking, explaining its strong performance on semantic metrics.

These results suggest that human attention patterns encode architecture-invariant cues of semantic importance. Regardless of whether information is processed autoregressively or bidirectionally, guiding attention toward regions that typically attract human gaze helps redistribute representational focus more effectively. The multimodal Gaussian formulation accommodates these differences without architectural redesign, demonstrating EyeLayer's flexibility and generalizability as a cognitively grounded attention module.

RQ3 Summary. EyeLayer generalizes well to encoder-only models, confirming that human attention patterns provide architecture-invariant cues for semantic importance and support flexible attention redistribution.

Table 2: CodeBERT performance with and without EyeLayer integration.

Model	BLEU-4	ROUGE-L	METEOR	BERTScore
CodeBERT	14.35	29.16	21.87	87.69
CodeBERT + EyeLayer	15.39 (**+1.04**)	30.70 (**+1.54**)	23.70 (**+1.83**)	88.30 (**+0.61**)

5.4 RQ4: Ablation Study on Multimodal Design

RQ4 investigates whether the multimodal Gaussian design, which models human attention through multiple distinct modes, provides advantages over simpler single mode alternatives.

To isolate the multimodal design's contribution, we implement a simplified EyeLayer variant that predicts attention using a single Gaussian distribution rather than a mixture. This architecture removes the sparse gating network and mode-specific prediction heads, and instead directly predicts the global centroid μ and spread σ from the code-level embedding, while retaining all other components, including low-rank perturbation, attention-guided weighting, and adaptive highway gating. We evaluate single-mode EyeLayer at early and late layer positions on Llama3.2-1B-instruct.

Table 3 shows that multimodal EyeLayer consistently outperforms single-mode variants across all metrics. Single-mode configurations show limited improvements over baseline, with early layer achieving minimal gains and late layer showing inconsistent performance. In contrast, multimodal EyeLayer delivers substantial improvements. For example, at layer 13, multimodal design achieves BLEU-4: 16.18 versus single-mode's 14.63 (+1.55), and METEOR: 29.33 versus 26.10 (+3.23), demonstrating clear advantages of modeling multiple attention modes.

The results support our hypothesis that human attention during code comprehension cannot be captured by a single Gaussian. A single-mode design can only represent one attention region, which forces a trade-off between narrow focus (small σ) and broad coverage (large σ), and thus fails to model multiple distinct areas of interest in complex functions. In contrast, the multimodal design enables sparse mode selection, where the gating network activates 1–3 modes adaptively based on code complexity. This allows the model to compose multiple attention patterns, such as scanning function signatures, following control flow, and inspecting implementation details. The substantial performance gains indicate that modeling diverse attention modes improves cognitive fidelity and justifies the added architectural complexity.

RQ4 Summary. Multimodal Gaussian design outperforms single-mode variants, demonstrating that modeling multiple attention modes better captures human gaze diversity and yields stronger semantic alignment.

6 Threats to Validity

There are two main threats to the validity of our work. First, our eye-tracking supervision derives exclusively from Java code comprehension, which may limit generalization to languages with different syntactic structures or paradigms. However, core code comprehension strategies are similar across languages. This implies that human attention patterns reflecting semantic importance may also transfer, but further validation is needed for EyeLayer across diverse programming languages. Second, our evaluation relies on automatic metrics that may not fully correlate with human-perceived summary quality or practical developer productivity in real-world scenarios. We mitigate this threat by employing four complementary metrics (BLEU, ROUGE-L, METEOR, BERTScore) spanning lexical overlap and semantic similarity dimensions, validating across diverse model architectures (decoder-only and encoder-only), and conducting qualitative analysis demonstrating meaningful semantic improvements in generated summaries. All experiments used fixed random seeds to ensure reproducibility and minimize bias.

7 Discussion and Future Work

Scaling Eye-Tracking Supervision. Our results show that 625 sparse eye-tracking samples provide consistent benefits, suggesting that scaling supervision through data augmentation or large-scale collection could further improve performance. Richer supervision would enable more expressive EyeLayer architectures capturing finer-grained attention patterns.

Richer Cognitive Signals. Our approach uses only static fixation—aggregated attention intensity. Eye-tracking can contain additional information: saccade patterns (revealing information-seeking strategies), and attention switches (capturing dynamic shifts in cognitive focus). Incorporating these temporal and sequential signals has the potential to provide richer supervision.

Generalization to Software Engineering Tasks. While we focus on code summarization, many SE tasks fundamentally involve code comprehension: bug localization, code review, and program repair all require identifying semantically important regions. Human attention patterns should transfer across tasks as developers employ similar cognitive strategies regardless of end goal. EyeLayer's effectiveness across both decoder-only and encoder-only architectures demonstrates its flexibility for integration into diverse models. However, future work should investigate whether attention patterns from code summarization tasks can transfer to other SE contexts, or whether collecting task-specific eye-tracking data yields stronger supervision signals.

Broader Implications. Beyond performance improvements, EyeLayer demonstrates grounding neural models in human cognitive processes rather than purely data-driven learning. This approach could enable more interpretable AI systems where models attend to code for reasons aligned with human reasoning, facilitating developer trust and effective human-AI collaboration as code intelligence tools become ubiquitous in development workflows.

8 Related Work

This section situates our work at the intersection of human-centered AI and automatic code summarization. We first review research that

Table 3: Ablation study comparing single-mode and multimodal EyeLayer designs on Llama3.2-1B.

Configuration	BLEU-4	ROUGE-L	METEOR	BERTScore
Baseline (SFT)	14.31	22.12	27.45	87.55
Single-mode (Early)	14.30	21.64	27.55	88.13
Single-mode (Late)	14.63	20.82	26.10	88.26
Multimodal (Late)	**16.18**	**23.51**	**29.33**	**88.51**

integrates cognitive and behavioral signals into software engineering models, emphasizing eye-tracking as a bridge between human and machine attention. We then discuss advances in code summarization, from transformer-based architectures to recent efforts incorporating human-like attention guidance.

8.1 Human-centered AI for Software Engineering

Human-centered AI for software engineering (SE) emphasizes aligning automated systems with human cognition and developer workflows. Empirical studies have shown that developers interact with AI assistants in complex ways: they often exhibit overconfidence while producing less secure code [36], alternate between acceleration and exploration modes depending on task certainty [6], and face persistent challenges in output validation and trust calibration [15, 26, 30]. Recent theoretical frameworks further characterize trust as a dynamic and multi-dimensional construct [9, 38], underscoring the need for models that are cognitively transparent and behaviorally adaptive.

Beyond behavioral analysis, recent research has sought to directly model cognitive processes underlying code comprehension. Early eye-tracking studies revealed that developer gaze patterns reflect semantic understanding during program reading [34, 37]. Building on this foundation, Bansal et al. [5] and Alakmeh et al. [2] predicted human attention from code structure and integrated gaze information to enhance summarization models. More recently, EyeTrans [55] and EyeMulator [54] incorporated gaze data into Transformer architectures, achieving measurable performance gains.

EyeLayer extends this research direction by being among the first to incorporate human cognitive signals into large language models. It leverages human attention as a transferable probabilistic prior, aiming for generalizable integration of human-like focus patterns across model architectures and tasks.

8.2 Automatic Code Summarization

The advent of large language models (LLMs) has catalyzed a paradigm shift in automatic code summarization, transitioning from traditional sequence-to-sequence architectures to transformer-based approaches that leverage extensive pre-training on code corpora. Early work such as Code2Seq [3] and retrieval-augmented methods [51] demonstrated that structural program representations and example-based retrieval can significantly enhance summary quality. The establishment of benchmarks like CodeXGLUE [28] standardized evaluation protocols and enabled systematic comparison across models and datasets. Building on these foundations, Shi et al. [43] identified key factors influencing neural summarization performance, while Gao et al. [16] and Fang et al. [12] explored in-context and prompt-based learning to adapt general-purpose LLMs for code summarization. Empirical studies further revealed that moderately sized, fine-tuned models can rival or surpass much larger general-purpose LLMs when supervision effectively captures task semantics [46], emphasizing the centrality of the fine-tuning process in code-oriented adaptation.

Recent work has focused on improving efficiency, robustness, and interpretability in LLM-based summarization [46]. Su et al. [45] applied knowledge distillation to reduce computational costs, while Mastropaolo et al. [31] proposed semantic-aware evaluation metrics to better assess summary fidelity. Virk et al. [47] exposed calibration deficiencies that undermine model reliability, and Mondal et al. [33] examined robustness to adversarial perturbations. Interpretability analyses further uncovered a persistent misalignment between model-generated attention and developer comprehension: Li et al. [25] showed that neural attention often diverges from code regions developers focus on, leading to summaries that are lexically fluent but semantically incomplete. This gap between surface-level correlations and true comprehension has motivated recent studies to augment fine-tuning with auxiliary behavioral cues such as eye-tracking, exemplified by EyeTrans [55], which guide transformer attention toward semantically salient regions.

EyeLayer continues this trajectory by strengthening the supervised fine-tuning of LLM-based summarization. Rather than redesigning model architectures or relying on heavy supervision, it introduces lightweight cognitive priors into the fine-tuning pipeline to steer attention toward functionally important code regions.

9 Conclusion

This work demonstrates that human cognitive patterns captured through eye-tracking can effectively enhance LLM-based code summarization. We introduced EyeLayer, a lightweight attention-augmentation module that integrates sparse human attention signals into LLMs through Multimodal Gaussian Mixture Models, enabling models to learn how developers naturally focus on semantically critical code regions during comprehension. Our evaluation across five models spanning different scales and architectures shows consistent improvements, validating that human expertise provides complementary signals that enhance LLM capabilities beyond what standard supervised fine-tuning achieves. Our methodology establishes a framework for incorporating human cognitive processes into LLMs for code comprehension, contributing to the development of more capable and interpretable developer tools as software systems continue to grow in complexity.

References

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A Transformer-based Approach for Source Code Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL).*

[2] Tarek Alakmeh, David Reich, Lena Jäger, and Thomas Fritz. 2024. Predicting Code Comprehension: A Novel Approach to Align Human Gaze with Code Using Deep Neural Networks. *Proceedings of the ACM on Software Engineering* 1, FSE (July 2024), 1982–2004. doi:10.1145/3660795

[3] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2019. code2seq: Generating Sequences from Structured Representations of Code. arXiv:1808.01400 [cs.LG] https://arxiv.org/abs/1808.01400

[4] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, Jade Goldstein, Alon Lavie, Chin-Yew Lin, and Clare Voss (Eds.). Association for Computational Linguistics, Ann Arbor, Michigan, 65–72.

[5] Aakash Bansal, Bonita Sharif, and Collin McMillan. 2023. Towards Modeling Human Attention from Eye Movements for Neural Source Code Summarization. *Proceedings of the ACM on Human-Computer Interaction* 7, ETRA (May 2023), 1–19. doi:10.1145/3591136

[6] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2022. Grounded Copilot: How Programmers Interact with Code-Generating Models. doi:10.48550/arXiv.2206.15000

[7] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *2015 IEEE 23rd International Conference on Program Comprehension.* IEEE, Florence, Italy, 255–265. doi:10.1109/ICPC.2015.36

[8] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. 2015. Attention-Based Models for Speech Recognition. In *Advances in Neural Information Processing Systems*, Vol. 28. Curran Associates, Inc.

[9] Rudrajit Choudhuri, Bianca Trinkenreich, Rahul Pandita, Eirini Kalliamvakou, Igor Steinmacher, Marco Gerosa, Christopher Sanchez, and Anita Sarma. 2024. What Guides Our Choices? Modeling Developers' Trust and Behavioral Intentions towards GenAI. doi:10.48550/arXiv.2409.04099

[10] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. 2019. On the Relationship between Self-Attention and Convolutional Layers. In *International Conference on Learning Representations.*

[11] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE).* 31–53. doi:10.1109/ICSE-FoSE59343.2023.00008

[12] Minying Fang, Xing Yuan, Yuying Li, Haojie Li, Chunrong Fang, and Junwei Du. 2025. Enhanced Prompting Framework for Code Summarization with Large Language Models. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA072 (June 2025), 24 pages. doi:10.1145/3728949

[13] Harshwardhan Fartale, Ashish Kattamuri, Rahul Raja, Arpita Vats, Ishita Prasad, and Akshata Kishore Moharir. 2025. Disentangling Recall and Reasoning in Transformer Models through Layer-Wise Attention and Activation Analysis. doi:10.48550/arXiv.2510.03366

[14] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages.

[15] Yujia Fu, Peng Liang, Amjed Tahir, Zengyang Li, Mojtaba Shahin, Jiaxin Yu, and Jinfu Chen. 2025. Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study. doi:10.48550/arXiv.2310.02059

[16] Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, and Michael R. Lyu. 2023. What Makes Good In-Context Demonstrations for Code Intelligence Tasks with LLMs?. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 761–773. doi:10.1109/ase56229.2023.00109

[17] Lisa Grabinger, Florian Hauser, Christian Wolff, and Jürgen Mottok. 2024. On Eye Tracking in Software Engineering. *SN Computer Science* 5, 6 (July 2024), 729. doi:10.1007/s42979-024-03045-3

[18] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, and et al. 2024. The Llama 3 Herd of Models. doi:10.48550/arXiv.2407.21783

[19] Alex Graves. 2014. Generating Sequences With Recurrent Neural Networks. doi:10.48550/arXiv.1308.0850

[20] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. 2015. DRAW: A Recurrent Neural Network For Image Generation. doi:10.48550/arXiv.1502.04623

[21] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620 [cs.SE] https://arxiv.org/abs/2308.10620

[22] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).

[23] Marcel Adam Just and Patricia A Carpenter. [n. d.]. A Theory of Reading: From Eye Fixations to Comprehension. ([n. d.]).

[24] Zachary Karas, Aakash Bansal, Yifan Zhang, Toby Li, Collin McMillan, and Yu Huang. 2024. A Tale of Two Comprehensions? Analyzing Student Programmer Attention during Code Summarization. *ACM Trans. Softw. Eng. Methodol.* 33, 7, Article 193 (Aug. 2024), 37 pages. doi:10.1145/3664808

[25] Jiliang Li, Yifan Zhang, Zachary Karas, Collin McMillan, Kevin Leach, and Yu Huang. 2024. Do Machines and Humans Focus on Similar Code? Exploring Explainability of Large Language Models in Code Summarization. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension* (Lisbon, Portugal) *(ICPC '24).* Association for Computing Machinery, New York, NY, USA, 47–51. doi:10.1145/3643916.3644434

[26] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2023. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. doi:10.48550/arXiv.2303.17125

[27] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out.* Association for Computational Linguistics, Barcelona, Spain, 74–81.

[28] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. (2021). arXiv:2102.04664 [cs.SE] https://arxiv.org/abs/2102.04664

[29] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. *CoRR* abs/2102.04664 (2021).

[30] Yunbo Lyu, Zhou Yang, Jieke Shi, Jianming Chang, Yue Liu, and David Lo. 2025. "my Productivity Is Boosted, but ..." Demystifying Users' Perception on AI Coding Assistants. doi:10.48550/arXiv.2508.12285

[31] Antonio Mastropaolo, Matteo Ciniselli, Massimiliano Di Penta, and Gabriele Bavota. 2024. Evaluating Code Summarization Techniques: A New Metric and an Empirical Characterization. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24).* Association for Computing Machinery, New York, NY, USA, Article 218, 13 pages. doi:10.1145/3597503.3639174

[32] Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. 2025. Talking Heads: Understanding Inter-Layer Communication in Transformer Language Models. doi:10.48550/arXiv.2406.09519

[33] Debanjan Mondal, Abhilasha Lodha, Ankita Sahoo, and Beena Kumari. 2023. Robust Code Summarization. In *Proceedings of the 1st GenBench Workshop on (Benchmarking) Generalisation in NLP*, Dieuwke Hupkes, Verna Dankers, Khuyagbaatar Batsuren, Koustuv Sinha, Amirhossein Kazemnejad, Christos Christodoulopoulos, Ryan Cotterell, and Elia Bruni (Eds.). Association for Computational Linguistics, Singapore, 65–75. doi:10.18653/v1/2023.genbench-1.5

[34] Matteo Paltenghi and Michael Pradel. 2021. Thinking like a Developer? Comparing the Attention of Humans with Neural Models of Code. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, Melbourne, Australia, 867–879. doi:10.1109/ase51524.2021.9678712

[35] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Pierre Isabelle, Eugene Charniak, and Dekang Lin (Eds.). Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 311–318. doi:10.3115/1073083.1073135

[36] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do Users Write More Insecure Code with AI Assistants?. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security.* ACM, Copenhagen Denmark, 2785–2799. doi:10.1145/3576915.3623157

[37] Paige Rodeghero, Collin McMillan, Paul W. McBurney, Nigel Bosch, and Sidney D'Mello. 2014. Improving Automated Source Code Summarization via an Eye-Tracking Study of Programmers. In *Proceedings of the 36th International Conference on Software Engineering.* ACM, Hyderabad India, 390–401. doi:10.1145/2568225.2568247

[38] Sadra Sabouri, Philipp Eibl, Xinyi Zhou, Morteza Ziyadi, Nenad Medvidovic, Lars Lindemann, and Souti Chattopadhyay. 2025. Trust Dynamics in AI-assisted Development: Definitions, Factors, and Implications. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE).* IEEE, Ottawa, ON, Canada, 1678–1690. doi:10.1109/ICSE55347.2025.00199

[39] Zohreh Sharafi, Timothy Shaffer, Bonita Sharif, and Yann-Gaël Guéhéneuc. 2015. Eye-Tracking Metrics in Software Engineering. In *2015 Asia-Pacific Software*

Engineering Conference (APSEC). 96–103. doi:10.1109/APSEC.2015.53

[40] Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. 2020. A Practical Guide on Conducting Eye Tracking Studies in Software Engineering. *Empirical Software Engineering* 25, 5 (Sept. 2020), 3128–3174. doi:10.1007/s10664-020-09829-4

[41] Zohreh Sharafi, Zéphyrin Soh, and Yann-Gaël Guéhéneuc. 2015. A Systematic Literature Review on the Usage of Eye-Tracking in Software Engineering. *Information and Software Technology* 67 (Nov. 2015), 79–107. doi:10.1016/j.infsof.2015.06.008

[42] Bonita Sharif and Jonathan I. Maletic. 2010. An Eye Tracking Study on camelCase and Under_score Identifier Styles. In *2010 IEEE 18th International Conference on Program Comprehension*. 196–205. doi:10.1109/ICPC.2010.41

[43] Ensheng Shi, Yanlin Wang, Lun Du, Junjie Chen, Shi Han, Hongyu Zhang, Dongmei Zhang, and Hongbin Sun. 2022. On the evaluation of neural code summarization. In *Proceedings of the 44th International Conference on Software Engineering*. ACM, 1597–1608. doi:10.1145/3510003.3510060

[44] Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. 2025. Layer by Layer: Uncovering Hidden Representations in Language Models. doi:10.48550/arXiv.2502.02013

[45] Chia-Yi Su and Collin McMillan. 2024. Distilled GPT for source code summarization. *Automated Software Engg.* 31, 1 (March 2024), 26 pages. doi:10.1007/s10515-024-00421-4

[46] Weisong Sun, Yun Miao, Yuekang Li, Hongyu Zhang, Chunrong Fang, Yi Liu, Gelei Deng, Yang Liu, and Zhenyu Chen. 2025. Source Code Summarization in the Era of Large Language Models. arXiv:2407.07959 [cs.SE] https://arxiv.org/abs/2407.07959

[47] Yuvraj Virk, Premkumar Devanbu, and Toufique Ahmed. 2025. Calibration of Large Language Models on Code Summarization. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE130 (June 2025), 21 pages. doi:10.1145/3729400

[48] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, and et al. 2025. Qwen3 Technical Report. doi:10.48550/arXiv.2505.09388

[49] Weiqiu You, Simeng Sun, and Mohit Iyyer. 2020. Hard-Coded Gaussian Attention for Neural Machine Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 7689–7700. doi:10.18653/v1/2020.acl-main.687

[50] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient Surgery for Multi-Task Learning. doi:10.48550/arXiv.2001.06782

[51] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. 2020. Retrieval-based neural source code summarization. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1385–1397. doi:10.1145/3377811.3380383

[52] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT. doi:10.48550/arXiv.1904.09675

[53] Yang Zhang, Yanfei Dong, and Kenji Kawaguchi. 2024. Investigating Layer Importance in Large Language Models. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, Miami, Florida, US, 469–479. doi:10.18653/v1/2024.blackboxnlp-1.29

[54] Yifan Zhang, Chen Huang, Yueke Zhang, Jiahao Zhang, Toby Jia-Jun Li, Collin McMillan, Kevin Leach, and Yu Huang. 2025. EyeMulator: Improving Code Language Models by Mimicking Human Visual Attention. doi:10.48550/arXiv.2508.16771

[55] Yifan Zhang, Jiliang Li, Zachary Karas, Aakash Bansal, Toby Jia-Jun Li, Collin McMillan, Kevin Leach, and Yu Huang. 2024. EyeTrans: Merging Human and Machine Attention for Neural Code Summarization. *Proceedings of the ACM on Software Engineering* 1, FSE (July 2024), 115–136. doi:10.1145/3643732