

A Supplementary Material

A.1 Broader Impact

The proposed Glance2Gaze advances vision-language models by introducing a cognitively inspired framework that harmonizes efficiency and performance through its two-stage attention mechanism. By mimicking human visual cognition—first capturing global context and then focusing on salient details—the method significantly reduces computational overhead while maintaining or improving accuracy across diverse tasks. This innovation broadens accessibility to advanced visual-language systems, enabling deployment in resource-limited settings such as edge devices or real-time applications. Its scalability to high-resolution and video inputs further extends practical utility in fields like medical imaging, autonomous systems, and multimedia analysis. Environmentally, the reduced computational demand aligns with sustainable AI development by lowering energy consumption. As a generalizable paradigm, Glance2Gaze also inspires future research in biologically inspired attention mechanisms, fostering interdisciplinary advancements in efficient multimodal learning.

A.2 Efficiency Analysis

Previous works [18, 21–23] typically measure efficiency by the computational cost of visual tokens in LLMs, a method we also adopt. Additionally, we analyze the extra computational load introduced by Glance Fusion.

FLOPs within Glance Fusion. Considering the computational cost incurred by Glance Fusion, we account for the FLOPs arising from all projection operations of both visual (Eq. 2) and textual (Eq. 3) inputs, as well as the overhead introduced by the visual-text correlation computation (Eq. 4). Specifically, the visual mapping introduces SNd_vd_t FLOPs, while the text mapping contributes SMd_t^2 FLOPs. Furthermore, the visual-text correlation involves S matrix multiplications, resulting in $SNMd_t$ FLOPs. Overall, the total FLOPs within the Glance Fusion module can be expressed as:

$$C_{glance} = SNd_vd_t + SMd_t^2 + SNMd_t. \quad (11)$$

FLOPs within LLM (Visual Tokens). We assess FLOPs of visual tokens within the LLM by examining two factors: (i) the standard forward pass through the decoder, and (ii) the additional cost introduced by the Gaze Compression module. For the visual token processing in Vicuna-7B [8], using a 3-layer MLP as FFN and 32 decoder layers, each layer incurs $4n_ld_t^2 + 2n_l^2d_t + 3n_ld_td_m$ FLOPs, where n_l and d_m denote the number of visual tokens and the hidden dimension of the FFN, respectively. In our framework, n_l is 576 before layer r and p_l thereafter, with l representing the LLM layer index. The FLOPs from Eq. 9 include 2 matrix multiplications and 4 linear projections, totaling $2p_{l-1}p_ld_t + 2p_{l-1}d_t^2 + 2p_ld_t^2$. Therefore, the total FLOPs within LLM are:

$$C_{llm} = \sum_{l=1}^R 4n_ld_t^2 + 2n_l^2d_t + 3n_ld_td_m + \sum_{l=r}^R 2(p_{l-1}p_ld_t + p_{l-1}d_t^2 + p_ld_t^2). \quad (12)$$

We found C_{glance} to be negligible compared to C_{llm} . For instance, in LLaVA-1.5-7B, with r set to 3 and p_r to 100, C_{llm} amounts to 1.1T FLOPs while C_{glance} is only 0.03T FLOPs, yielding a proportion of 0.0272. Given its insignificance, C_{glance} can be omitted from detailed computational analysis, allowing us to concentrate on the more impactful C_{llm} for performance assessment.

A.3 More Implementation Details

A.3.1 Image Understanding Tasks

Training Recipes. Following LLaVA [3], we employ a two-stage training strategy for Glance2Gaze. In the first stage, we align image-text pairs by retaining the LLaVA architecture and training only the projector on the LLaVA-558K dataset for one epoch with a batch size of 256 and a learning rate of $1e-3$. In the second stage, we incorporate Glance Fusion and Gaze Compression into LLaVA, training all parameters except the visual encoder for one epoch with a batch size of 128 and a learning rate of $2e-5$.

To train LLaVA-NeXT-7B [15], we employ Open-LLaVA-NeXT [53], an open-source replication, due to proprietary restrictions on the original code and training sets. Our approach involves a two-stage

Table 7: Compression configurations employed to LLaVA-1.5-7B.

FLOPs	r	P
33%	9	[256, 209, 170, 139, 114, 93, 76, 62, 50, 41, 33, 27, 22, 18, 15, 12, 10, 8, 6, 5, 4, 3, 2, 2]
22%	7	[121, 103, 88, 75, 64, 54, 46, 40, 34, 29, 24, 21, 18, 15, 13, 11, 9, 8, 7, 6, 5, 4, 3, 3, 2, 2]
11%	3	[100, 87, 77, 67, 59, 52, 45, 40, 35, 30, 27, 23, 20, 18, 16, 14, 12, 10, 9, 8, 7, 6, 5, 4, 4, 3, 3, 2, 2, 2]

Table 8: Compression configurations employed to LLaVA-NeXT-7B.

FLOPs	r	P
22%	7	[121, 103, 88, 75, 64, 54, 46, 40, 34, 29, 24, 21, 18, 15, 13, 11, 9, 8, 7, 6, 5, 4, 3, 3, 2, 2]
11%	3	[100, 87, 77, 67, 59, 52, 45, 40, 35, 30, 27, 23, 20, 18, 16, 14, 12, 10, 9, 8, 7, 6, 5, 4, 4, 3, 3, 2, 2, 2]
5%	2	[25, 23, 21, 19, 18, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 7, 6, 6, 5, 5, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2]

training process. Similar to LLaVA-1.5-7B, the proposed Glance2Gaze is integrated only in the second stage. In the first stage, we focus solely on training the projector with a batch size of 256 and a learning rate of $1e-3$ for one epoch using the LLaVA-558K dataset. In the second stage, Glance2Gaze is embedded into the LLaVA-NeXT architecture, with all parameters including the visual encoder unfrozen for training over two epochs with a batch size of 128 and a learning rate of $2e-5$, utilizing the same dataset as in [53].

Implementation Details. In Glance Fusion, \mathbb{L} is consistently set to $\{7, 13, 19, 23\}$ to capture fine-detailed features. For Gaze Compression, we adjust r and p_r to manage different computation configurations, as detailed in Table 7 and 8.

LLaVA-NeXT processes high-resolution images by dividing them into multiple sub-images (up to 5) for individual handling by the visual encoder, which are subsequently concatenated within the LLM. Both Glance Fusion and Gaze Compression are applied to each sub-image independently.

A.3.2 Video Understanding Tasks

Training Recipes. Following Video-LLaVA [34], we employ a two-stage training strategy. In the first stage, only the projector is trained for one epoch with a batch size of 256 and a learning rate of $1e-3$, using the LLaVA-558K dataset and a subset of Valley [58], while retaining the original architecture. In the second stage, we integrate Glance Fusion and Gaze Compression into Video-LLaVA, freeze the vision encoder, and finetune all other parameters with a batch size of 128 and a learning rate of $2e-5$, using the LLaVA-665K dataset and 100k video-text instructions from Video-ChatGPT [34].

Implementation Details. In Video-LLaVA [34], 8 frames are sampled from each video, with each frame encoded into 256 tokens, resulting in 2048 tokens per video. For Glance Fusion, \mathbb{L} is set to $\{7, 13, 19, 23\}$ by default. Gaze Compression is applied to each frame, with $r = 3$ and $p_r = 10$, yielding a model that operates at 6% of the FLOPs required for full token retention.

A.3.3 More Ablation Studies

More ablation on the effect of a shared query pool. Figure 5 presents additional ablation results comparing layer-specific query embeddings with a shared query pool in Gaze Compression. While the layer-specific approach increases parameters, it significantly degrades performance across all datasets.

More ablation on the effect of Glance Fusion. To assess the importance of global glance fusion prior to gaze compression, we compare Gaze Compression alone with the full Glance2Gaze in

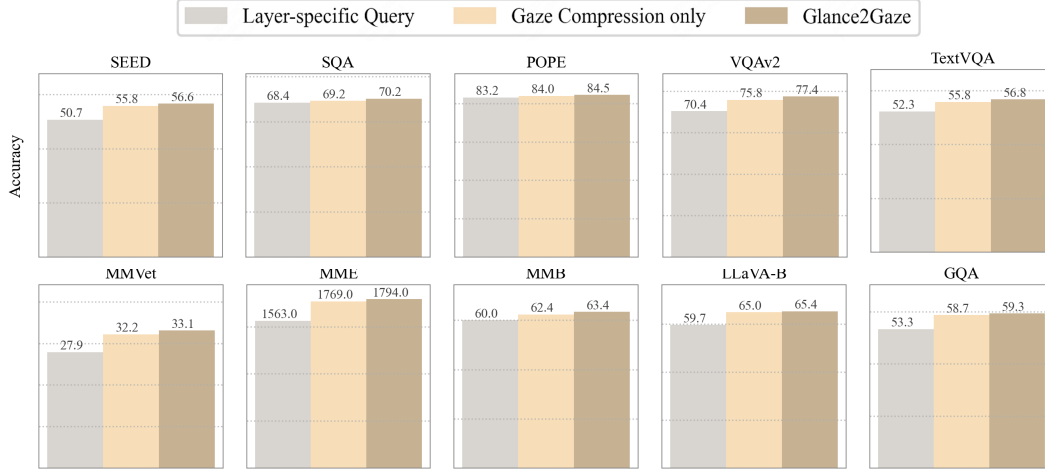


Figure 5: More ablation results on significance of shared query pool and the effect of Glance Fusion.

Table 9: Ablation studies on r and p_r , evaluated on VQAv2 dataset.

Method	$p_r=256$					$r=9$			
	$r=1$	$r=3$	$r=9$	$r=17$	$r=25$	$p_r=32$	$p_r=128$	$p_r=256$	$p_r=529$
FLOPs	12%	17%	33%	56%	78%	24%	28%	33%	42%
Acc	70.2	74.6	77.6	78.1	78.4	74.7	75.4	77.6	77.9

Figure 5. Adding Glance Fusion before Gaze Compression significantly improves performance across all datasets, with notable gains on fine-grained OCR tasks like TextVQA and VQAv2.

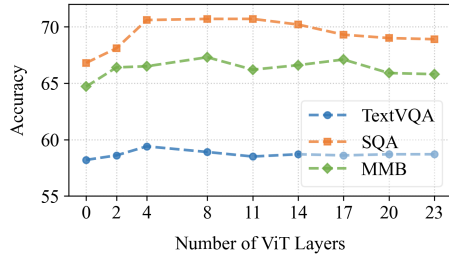
Ablation study on r and p_r . In Gaze Compression, the initial compression layer index r and the size of query embedding p_r together shape the compression ratio. To evaluate the performance impact of each variable independently, we varied one while holding the other constant, facilitating a comparative analysis of their effects, as shown in Table 9. It reveals a direct proportional relationship between accuracy and the starting compression layer when p_r is constant; larger r corresponds to higher accuracy. This is logical, as premature compression may prevent the LLM from fully processing visual tokens. Similarly, opting for a larger query embedding size yields benefits, as it allows richer information capture during compression.

Exploration on number of layers in Glance Fusion. We fuse different numbers of ViT layers, with results displayed in Figure 6 (a). Interestingly, adding more ViT layers does not always result in proportional performance gains. This aligns with observations in Figure 1, suggesting significant redundancy across layers and certain intermediate layers in ViT not contributing to instruction comprehension. To balance performance across all datasets, we selected 4 layers for fusion in this study.

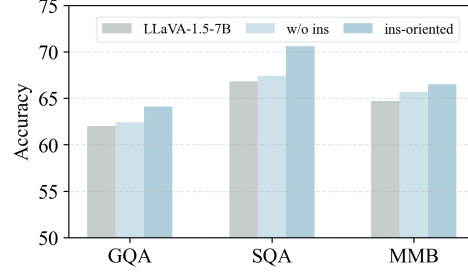
Necessity of Instruction in Glance Fusion. To demonstrate the significance of instruction in Glance Fuse, we compared it with straightforward feature averaging across layers. As illustrated in Figure 6 (b), indiscriminate fusion of ViT layers results in minimal improvement, whereas instruction-guided fusion significantly enhances features and boosts performance.

A.3.4 More efficiency comparison

We provide comprehensive tables summarizing the training cost, inference latency, prefilling time, and throughput across different frameworks and compression configurations, offering a clear overview of the computational efficiency, as shown in Table 10.



(a) Exploration on number of ViT layers



(b) Importance of Instruction-oriented integration

Figure 6: Ablation studies on Glance Fusion module.

Table 10: Comparison of computational efficiency between Glance2Gaze and baseline models. Metrics include training GPU hours, inference TFLOPs, latency, CUDA time, and throughput.

Method	Training GPU Hours	Inference TFLOPs	Latency (ms)	CUDA Time (ms)	Throughput
<i>LLaVA-1.5-7B family</i>					
LLaVA-1.5-7B	104	10.07	185.9	115	28.754
Glance2Gaze (33% FLOPs)	72	3.36	133.2	66	40.022
Glance2Gaze (22% FLOPs)	60	2.22	109.2	32	51.351
Glance2Gaze (11% FLOPs)	43	1.11	100.2	22	56.820
<i>LLaVA-Next-7B family</i>					
LLaVA-Next-7B	366	53.83	475.2	313	10.884
Glance2Gaze (22% FLOPs)	242	11.84	262.0	100	19.741
Glance2Gaze (11% FLOPs)	173	5.53	191.9	90	30.719
Glance2Gaze (5% FLOPs)	87	2.67	151.2	49	34.121
<i>Video-LLaVA family</i>					
Video-LLaVA	297	37.38	739.6	342.2	21.599
Glance2Gaze (6% FLOPs)	151	2.24	484.3	125.1	35.095