A. Experimental Setup in Detail

We describe the experimental setup used to evaluate our input-adaptive inference mechanism in detail. We implemented our strategy on top of the codebases provided by the authors of HAMT [10], DUET [11], and VLN-CE↻BERT [35]. During inference, instead of using cached image features, we integrate the original encoder (ViT-B/16 [16] for HAMT and DUET and ResNet-152 [26] for VLN-CE↻BERT) to process the images directly.

Hardware and software. We run our experiments on a machine equipped with an Intel Xeon processor with 48 cores, 64GB of DRAM, and 8 NVIDIA A40 GPUs, with all inference tasks performed on a single GPU with a batch size of 1. Following the original HAMT study, we use Python, PyTorch, and Cuda for all experiments, with versions in accordance with the original studies [10, 11, 35]. For GFLOPs calculations, we use the Python library thop[1].

Datasets. We describe the benchmarks we use in detail:

- **R2R** [1] is based on Matterport3D [7], containing 10,567 panorama views taken from 90 photo-realistic houses. The dataset includes 7,189 shortest-path trajectories, each of which is associated with 3 natural language instructions. The training, validation (seen), validation (unseen), and test (unseen) sets include 61, 56, 11, and 18 houses, respectively. The validation (seen) set consists of houses in the training set, used to check the generalization status of a model during training, while the sets marked as 'unseen' are the houses not in the training set.
- **R2R-Back** [10] requires the agent to return to its starting point after reaching the destination. To complete the task, the agent must remember its navigation history. A return command is appended to each R2R instruction, and the reversed path is provided as guidance for the return trip.
- **R2R-Last** [10] uses only the last sentence from the original R2R instructions to describe the destination.
- **REVERIE** [47] provides high-level instructions, closer to those given by humans, replacing the step-by-step instructions of R2R. Instead of navigating to a target location, the agent is required to identify and localize the target object upon arrival, making the task more complex and realistic. The dataset includes 4,140 target objects, which are categorized into 489 distinct groups.
- **CVDN** [53] requires the agent to navigate based on long, potentially unclear instructions. The agent interacts with a navigator through question and answer dialog to clarify and complete the task. In total, it has 2,050 human-human navigation dialogues, consisting of over 7,000 navigation trajectories accompanied by question-answer interactions, covering 83 matterport3D houses.
- **SOON** [66] is similar to REVERIE but contains longer and more detailed instructions. The average length of these instructions is 47 words, with path lengths varying from 2 to 21 steps. It requires the agent to navigate by understanding the relationship between objects in the environment to accurately locate the target object.
- **R2R-CE** [36] is a continuous version of R2R supported by the Habitat simulator. To generate the dataset, Krantz *et al.* convert the static panoramic scene data in Matterport3D into a continuous environment using mesh-based 3D reconstructions. R2R trajectories are then transferred by mapping their nodes to the closest navigable locations on the reconstructed mesh. Non-navigable nodes (e.g., those placed on furniture or spanning disjoint regions) were filtered out. The final dataset consists of 4475 successfully transferred trajectories, each paired with the original R2R instruction set. Note that unlike the original R2R setting, where agents teleport between nodes, R2R-CE requires agents to navigate using low-level actions such as moving forward and turning.

[1] https://pypi.org/project/thop

B. Optimal Hyperparameters for Adapting MuE

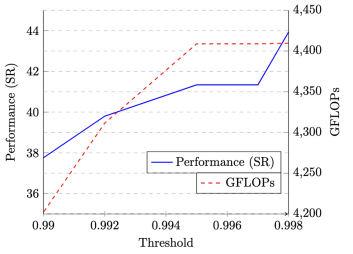

Figure 6. **Comparison of performance (in SR) and GFLOPs in MuE across different thresholds.**

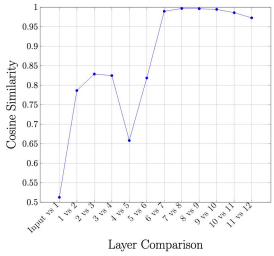

Figure 7. **Cosine similarity between adjacent layers of ViT used in HAMT.**

To best evaluate MuE on VLN tasks, we perform a hyperparameter sweep over the early-exit threshold. Figure 6 presents the performance (in SR) and GFLOPs across different early exit thresholds applied to the MuE version of ViT used in the HAMT agent, tested on the R2R dataset. The lowest threshold we report is 0.99, as lower thresholds caused a dramatic drop in performance (more than 50%). As the threshold increases, the success rate of the MuE agent increases substantially but at the cost of computational savings. Even for thresholds close to 1, meaning that the ViT is using a majority of its layers for each input, we still see a large performance drop compared to the baseline agent. As we discuss in Sec 3.2, this is likely because MuE statically applies early-exits, causing it to under-process important components of the panorama such as navigable views.

Why does MuE underprocess important views? The intuition behind MuE [51] is that the activations of Transformer-based vision models *saturate*, where their similarity between

layers peaks early on, and is maintained at future stages of computation, suggesting a lack of new/useful information. MuE then exploits this property to skip the later layers without a significant loss in performance. So, for MuE to be successful, the similarity of activations must sufficiently saturate and not decrease at later layers. However, as shown in Figure 7, the necessary saturation pattern is not observed in the VLN setting. The cosine similarity peaks between layers 7 and 8 but then decreases for all future layers. This explains the significant performance drop when MuE is directly applied to VLN agents, as it consistently early-exits despite saturation not being achieved.

Algorithm 2 SimHash Algorithm

Input: a current view v_i
Output: a binary hash key
function HASH(v_i)
  $key \leftarrow \varnothing$
  for each hp **in** Hyperplanes **do**
    $sign \leftarrow \texttt{DotProduct}(hp, v_i)$
    $hash\_val \leftarrow (sign > 0)$ ▷ converts to binary
    $key \leftarrow key + hash\_val$
  end for
  return key
end function
Input: a hash table h, a current view v_i, an embedding e_i
Output: a hash table h
function ADDTOHASHTABLE(h, v_i, e_i)
  $key \leftarrow \texttt{Hash}(v_i)$
  $h \leftarrow \texttt{InsertToHashTable}(key, v_i, e_i)$
  return h
end function
Input: a hash table h, a current view v_i
Output: an embedding e_i
function FINDSIMILAR(h, v_i)
  $s_{max} \leftarrow -1$
  $key \leftarrow \texttt{Hash}(v_i)$
  $bucket \leftarrow h.get(key)$
  for each $(v_{candidate}, e_{candidate})$ **in** bucket **do**
    $s \leftarrow$ **CosineSimilarity**(v_i, $v_{candidate}$)
    if $s > s_{max}$ **then**
      $s_{max} \leftarrow s$
      $e_{best} \leftarrow e_{candidate}$
    end if
  end for
  if $s_{max} > threshold$ **then**
    $e_i \leftarrow e_{best}$
  else
    $e_i \leftarrow \varnothing$
  end if
  return e_i
end function

C. Our LSH Algorithm in Detail

A core mechanism we introduce in Sec 3.2.3 is our SimHash algorithm, used to avoid reprocessing previously seen and similar images. Algorithm 2 details our implementation.

(line 1-9) Hashing RGB vectors. Given an image, we first hash the raw RGB vector into a short binary encoding using random projection [3, 8]. The algorithm calculates the dot product between the image vector and each hyperplane. If the dot product is positive, it assigns a binary value of 1, otherwise it assigns 0. These binary values are sequentially appended to form a complete binary hash key. The length of the hash key is determined by the number of hyperplanes used in the projection.

(line 10-14) Adding embeddings to the hash table. This function is used to insert processed images and their corresponding embeddings into the hash table for future use.

(line 15-32) Retrieving a similar embedding. This function takes an image we have not yet processed and tries to find a suitable embedding candidate. We first obtain all embeddings with images similar to the current image by hashing it into its binary encoding and accessing the corresponding bucket in the hash table. We then loop through all images associated with the similar embeddings and find the one yielding the highest similarity score (in our main experiments, the score is computed using cosine similarity). If this score exceeds our threshold hyperparameter, we return the associated embedding; otherwise, we return nothing.

Running the algorithm. We employ the above three functions to run SimHash on an arbitrary panorama. For each extended navigable view (other views are omitted and explained in Algorithm 1), we attempt to use a high-similarity embedding from the hash table. If it exists, we reuse this embedding for the current view and continue to the next. If not, we need to process the view using the ViT adapted for MuE, and then add the image and its embedding to the hash table. After processing the entire panorama, we return the set of final embeddings to be used for agent navigation.

Storage overhead analysis. Here, we consider the storage overhead necessary to deploy our hashing algorithm on VLN agents. Our LSH technique stores pairs of images and embeddings. In the benchmarks we consider, these images are of size 3x224x224 (Matterport3D) or 3x480x640 (Habitat). The embedding size depends on the model: 197x768 for HAMT and DUET (the number of ViT patches times the model's hidden dimension) and 2048 for VLN-CE↻BERT (the hidden dimension of ResNet-152). These are stored in full-precision floating-point format (4 bytes per value), resulting in $(3 \times 224 \times 224 + 197 \times 768) \times 4 \approx 1.2$ MB for HAMT and DUET and $(3 \times 480 \times 640 + 2048) \times 4 \approx 3.7$ MB for VLN-CE↻BERT per cached pair. For standard VLN, the longest navigation route was ∼12 steps (from R2R-Back). Assuming caching of all 36 images per panorama, the worst-case storage overhead is 522.7 MB. However, in practice,

most tasks involve 5–7 steps, and we cache at most 14 images per step, yielding a more typical overhead of 84.7–118.6 MB. For continuous VLN, the longest navigation route is ∼130 steps, and we cache at most 6 views, leading to a worst-case overhead of 2.9 GB. The average trajectory length is 56 steps, with about 3 views cached per step, resulting in an average overhead of 609.6 MB. Given that modern DRAM sizes are orders of magnitude larger, this storage overhead remains manageable for practical deployment.

D. Full Standard VLN Evaluation Results

Agent	Task	Method	Performance					GFLOPs
			TL	OSR	SR	SPL	GP	
HAMT	**R2R**	Base	11.53	74.29	66.16	61.49	-	4763.24
		Ours (All)	12.87	71.95	60.41	54.50	-	1917.61
	R2R-Back	Base	20.56	-	55.43	52.34	-	8181.55
		Ours (All)	20.53	-	49.21	46.47	-	3331.80
	R2R-Last	Base	12.28	54.24	47.85	42.27	-	4982.68
		Ours (All)	12.36	49.72	41.93	36.97	-	2589.44
	CVDN	Base	-	-	-	-	4.88	11022.03
		Ours (All)	-	-	-	-	4.45	4773.34
DUET	**R2R**	Base	13.94	81.10	71.73	60.57	-	4998.00
		Ours (All)	14.21	73.86	63.47	52.35	-	2026.30
	SOON	Base	35.87	50.38	36.19	22.67	-	9997.81+C
		Ours (All)	42.36	54.22	36.43	20.37	-	4533.83+C

Table 9. **Performance and efficiency of the baseline agents versus our improved-efficiency agents across multiple benchmarks.** We denote the cost of object feature extraction as C.

Table 9 complements our main evaluation of standard VLN in Sec 4.1 with additional benchmarks: R2R [1], R2R-Back [10], R2R-Last [10], CVDN [53], and SOON [66]. For CVDN, we report the additional evaluation metric Goal Progress (GP), which assigns a higher score as the agent moves closer to the goal, indicating better performance [10].

The upper section of the table compares the performance and efficiency of the baseline and our efficient HAMT agents. For R2R and R2R-Back, our strategy reduces computations by 60% with an SR drop of 9–11%. For R2R-Last, we reduce computation by 48%, with a 12% reduction in SR. Finally, for the CVDN evaluation, our efficient model reduces computation by 57%, with only a 9% decrease in GP.

The lower section of the table presents a comparison of the performance and efficiencies of the DUET agents. For R2R, our strategy achieves a 59% speed-up with a 12% decrease in SR. For SOON, we observed a marginal increase in SR accompanied by a 10% drop in SPL, while saving 5463.98 GFLOPs (a 55% reduction in visual feature processing). These results demonstrate that our efficiency strategies are applicable across different benchmarks, achieving substantial computational savings while maintaining an acceptable trade-off in performance.

Robustness to navigation length. It is possible that the errors introduced by our method *propagate*, resulting in worse agent navigation for longer trajectories. We study if this is the case by considering the *navigation error* (NE)—the distance of an agent's final position to the target position (in meters)—on benchmarks with varying path lengths. We deploy all of our proposed methods (simultaneously) on the HAMT agent and report the changes in NE and GFLOPs compared to the baseline in Table 10.

Agent	Task	Average Path Length	ΔNE(↓)	ΔGFLOPs(↓)
HAMT	**R2R**	6.0	+0.53	-2845.63
	R2R-Last	6.0	+0.45	-2393.24
	R2R-Back	12.0	+0.54	-5463.98
DUET	**R2R**	6.0	+0.68	-2971.70
	SOON	9.6	-0.44	-5463.98

Table 10. **Performance of our efficient HAMT agent on benchmarks with different path lengths.** ΔNE and ΔGFLOPs are the changes in navigation error (NE) and GFLOPs compared to the baseline agent. The path length is the minimum number of navigation actions needed to reach the target destination.

We find our method is largely robust to longer path lengths. The NE does not increase for longer trajectories, and we even see a decrease for the SOON benchmark, which has an average path length 3.6 more steps than R2R. The results also show that our efficient VLN agent sees roughly proportional computational savings for longer paths. For example, the average path length in R2R-Back is double R2R, and we achieve a 1.92x larger reduction in GFLOPs for the HAMT agent.

Task	Agent	Method	Wall-time (s)
R2R	HAMT	Base	200811
		Ours	119514
	DUET	Base	268962
		Ours	170464

Table 11. **Wall-time comparison** between the baseline agent and our efficient agent on the R2R task.

Runtime comparison. To validate that our approach improves efficiency in the real world, we report the wall-time comparison between our efficient VLN model and the baseline VLN for both HAMT and DUET agents, tested on the R2R validation unseen split, in Table 11. Evidently, our efficient strategy applied to the VLN agents results in significant runtime savings, with an approximate 40% reduction. It is important to note that the disparity between the 60% GFLOPs savings and the 40% runtime reduction can be attributed to various hardware and software-related factors, such as simulation overhead, memory bandwidth limitations, or cache latency.

Method	TL(↓)	OSR(↑)	SR(↑)	SPL(↑)	GFLOPs(↓)
None (Base)	11.53	74.29	66.16	61.49	4763.24
k-extension	12.52	71.86	61.30	55.79	2,408.99
thresholds	12.33	72.46	62.62	57.39	3,867.46
LSH	11.53	74.20	66.11	61.47	3,894.76
k-extension+LSH	12.52	71.90	61.17	55.63	2,013.48
k-extension+thresholds	12.89	71.95	60.41	54.57	2,294.23
thresholds+LSH	12.33	72.41	62.49	57.33	3,190.66
All	12.87	71.95	60.41	54.50	1,917.61

Table 12. **Performance of all combinations of our speed-up techniques** (k-extensions, early-exiting, and LSH) with the HAMT agent on the R2R benchmark.

E. Per-Mechanism Analysis

In most experiments, we treat our proposed mechanisms as a single unit by applying all three simultaneously. While this is the most flexible and offers the best trade-off between performance and efficiency, analyzing each mechanism independently can provide valuable insights into its concise impact. Here, we present results on a per-mechanism basis.

Effectiveness. In Sec 4.1, we apply our k-extension technique and then add adaptive thresholding early-exiting (denoted thresholds in Table 3) and locality-sensitive hashing (LSH) as we found those combinations of techniques offer the most computational savings. Here, we study all combinations of three efficiency mechanisms. To use early-exiting and LSH without k-extension, we treat every non-navigable view as one that can be early-exited or hashed. Navigable views are still fully processed. We report results for the HAMT agent on the R2R benchmark in Table 12.

The results show that between individual techniques, k-extension offers the best computational savings with a 49% reduction compared to the baseline agent. Early-exiting and LSH only reduce GFLOPs by $\sim$18% because early-exiting still requires processing every view, and LSH reuses only a minority of cached image embeddings. We find that LSH provides better performance than the other two individual mechanisms, with an SR only 0.05 lower than the baseline. This is likely because the cached embeddings reused by LSH are near-identical, having a negligible impact on performance when interchanged. However, it is far less efficient than when combined with our other techniques.

The combination we do not present in Table 3, early-exiting and LSH (**thresholds+LSH**), provides slightly better performance than combinations using k-extension but at the cost of 39–66% more GFLOPs. This suggests that retaining and partially processing/reusing the non-navigable views mitigates performance drop but is not nearly as efficient as k-extension. Overall, we find that all combinations of our techniques fare well, offering different trade-offs between performance and efficiency.

Robustness to natural corruptions. Now, we complement Sec 4.4 and study the robustness of each of our proposed mechanisms to visual corruption. We select the Low Lighting and Motion Blur corruptions based on their varying impact on performance and being more likely to occur in real-world VLN systems. We apply our methods to the HAMT agent and report results on R2R in Table 13.

Our methods appear more robust to Low Lighting than Motion Blur, which corroborates our findings in Sec 4.4. Across both corruptions, k-extension and early-exiting see a slight increase of 150–200 GFLOPs compared to the results in Table 12. This can likely be attributed to the increased trajectory length, and for early-exiting, we also find that the OOD samples require more ViT layers before sufficiently saturating. Both mechanisms result in significant drops in performance, though less than when we apply all simultaneously (results shown in Table 8). Early-exiting is slightly more robust, achieving a 2–7% higher SR, which makes sense as it processes strictly more images than k-extension.

Interestingly, LSH functions extremely well when Low Lighting is applied. It offers a $\sim$49% reduction in GFLOPs, compared to just 18% when no corruption is present. We hypothesize that the reduced lighting makes more images similar, causing our algorithm to find more matches and reuse more embeddings. It also offers significant robustness, only incurring a 1% point drop in SR. It seems like our caching mechanism is better suited for this environment, a finding we hope to explore in future work. For Motion Blur, LSH is less successful, being more robust than our other mechanisms but with minimal computational savings.

Corruption	Method	TL(↓)	OSR(↑)	SR(↑)	SPL(↑)	GFLOPs(↓)
Low Lighting	None (Base)	12.15	71.31	62.58	57.23	4903.06
	k-extension	13.86	71.14	57.34	50.78	2571.06
	thresholds	13.63	70.29	58.79	52.16	4099.21
	LSH	12.95	71.43	61.47	55.19	2444.05
Motion Blur	None (Base)	12.41	68.20	59.13	54.01	4996.64
	k-extension	14.03	65.13	53.77	48.01	2588.06
	thresholds	13.81	68.20	57.51	51.05	4073.04
	LSH	12.39	68.03	59.30	54.04	4030.52

Table 13. **Performance under visual corruption of our methods applied *independently*** to the HAMT agent on the R2R benchmark.

F. Information Loss Analysis

In this section, we explore what types of information are lost when applying each of our speed-up techniques.

Multi-exiting with thresholds. To assess the effect of processing views through fewer ViT layers, we analyze attention visualizations. Figure 8 illustrates attention maps from our efficient HAMT agent on a representative view. In this example, the correct action is to ignore the bathroom and move to the side. As the exit layer decreases, HAMT focuses more on the bathroom, indicating a slight degradation in visual

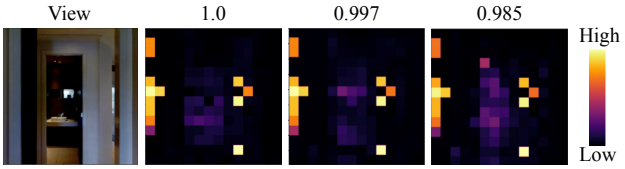

Figure 8. **Attention visualization** across different exit thresholds on HAMT. Lower thresholds use fewer ViT layers.

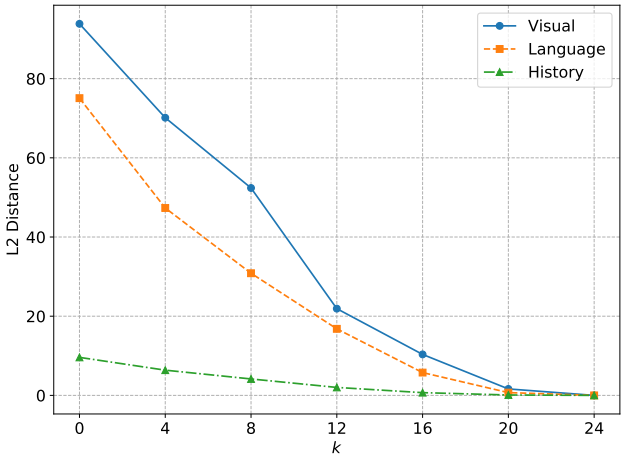

Figure 9. **L2 distance of cross-modal embeddings** from the baseline and our efficient HAMT agent for different k values. Embeddings are computed on 100 navigation instructions from R2R.

understanding. However, this change is minimal, and the overall navigation outcome is unaffected. Therefore, our adaptive thresholding technique provides an effective trade-off between computational efficiency and visual fidelity.

k-extensions. For k-extensions, we fully mask non-navigable views, making local attention visualizations uninformative. To capture the *global* impact of this masking, we measure the change in embeddings after processing through the cross-modal transformer. Specifically, we extract visual, language, and history embeddings from 100 navigation steps of HAMT on R2R. We then apply the k-extensions technique, re-extract the embeddings, and compute the mean L2 distance for each navigation step. To ensure comparability, we only consider the first step in each environment, as subsequent steps may diverge.

Figure 9 shows the results across different values of k. As k increases (i.e., more views are processed), the L2 distance for all embedding types decreases significantly. Interestingly, while $k = 4$–6 only marginally reduces these distances, we still observe strong performance in Sec. 4.1. This suggests that much of the information captured by these embeddings is not critical for navigation—an insight we leverage for computational efficiency. Masking views affects visual embeddings the most, as they consistently have the highest L2 distance for all values of k. However, we observe that

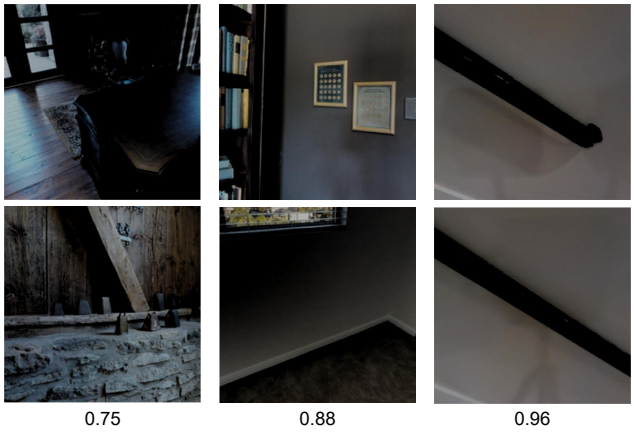

Figure 10. **Cosine similarity of different views** from R2R. Comparisons are made between the upper and lower images.

language embeddings, which encode the navigation instructions, are also notably impacted. This further explains the performance degradation: if the agent does not understand the instruction, it may fail to navigate or stop appropriately. In contrast, history embeddings are more resilient, likely because we only evaluate the first navigation step where historical context is minimal. Overall, these results indicate that masking views leads to information loss that extends beyond visual perception. However, this loss is not critical for effective navigation with the appropriate choice of k.

LSH. Finally, for LSH, we analyze what types of semantic information are lost when replacing embeddings by comparing images with different cosine similarities. Figure 10 shows three representative examples. When the cosine similarity is low (<0.85), the views typically depict entirely different scenes or locations (e.g., the left pair of views). Reusing the corresponding embeddings in this case would result in a complete loss of information, substantially degrading agent performance. In contrast, when the cosine similarity is above 0.85—the threshold used in Sec 4.1—the views are generally much more semantically similar. For instance, the middle pair of views both show a wall of similar color, while the right pair depicts a slightly shifted angle of the same handrail. The embeddings of such images likely encode similar information with minimal loss, which explains the limited impact of our LSH technique on performance.

G. Similarity Metrics Comparison

Other than the three similarity metrics we use in Sec 4.3, we test three additional metrics for comparison: SURF [5], SIFT [41], and ORB [48]. These are feature detection and description algorithms designed to identify and match keypoints in images. The similarity scores are computed by dividing the number of matching keypoints by the minimum number of keypoints detected in the two images. We test all six algorithms on two sets of scenes, reflecting shifts caused

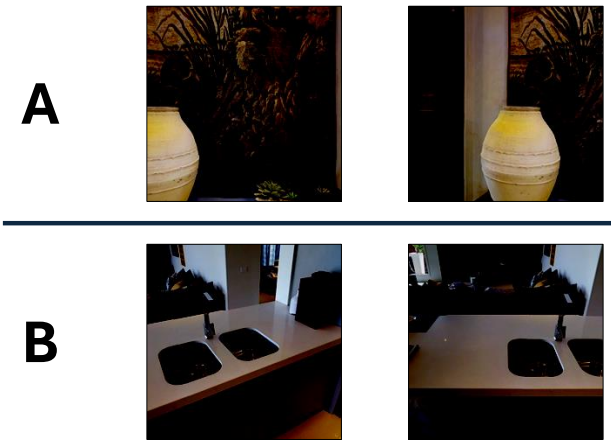

Table 14. **Two sets of example views (A and B)** demonstrating non-identical but similar views that have been slightly shifted during navigation.

Simiarlity Metrics	Set A	Set B
SSIM [57]	0.24	0.32
FSIM [63]	0.26	0.27
LPIPS [65]	0.55	0.62
SURF [5]	0.31	0.32
SIFT [41]	0.45	0.37
ORB [48]	0.07	0.19

Table 15. **Similarity scores measured on Set A and B.** We test 6 different similarity metrics.

by an agent's changing perspectives during navigation.

Figure 14 illustrates the two scenes, and Table 15 summarizes the quantitative comparison. Among the three metrics we employ for our main evaluation, LPIPS demonstrates a higher similarity measure of approximately 60% for both sets. In contrast, SSIM and FSIM are less effective at capturing the similarity between views in Sets A and B. The three additional metrics (SURF, SIFT, and ORB) are also ineffective in providing reliable similarity scores for both image sets A and B. Our qualitative comparison of different similarity metrics applied to sets of similar scenes highlights the challenges in accurately identifying true visual similarity. We believe that an accurate measure of scene similarity is crucial for further reducing the computational demands of a VLN agent, and we leave this for future work.

H. Performance-Efficiency Trade-off Analysis

In order to illustrate our tunable performance-efficiency trade-off, we show that even when limiting the performance drop to under 5%, our input adaptive inference method applied to the HAMT agent achieves significant computational savings. For reference, the baseline HAMT model achieves a SR of 66.16 with a computational cost of 4763.24 GFLOPs. Figure 11 shows that with a 3–5% drop in SR, we still manage to achieve 43–50% savings in GFLOPs. These results were tested on the R2R validation unseen dataset.

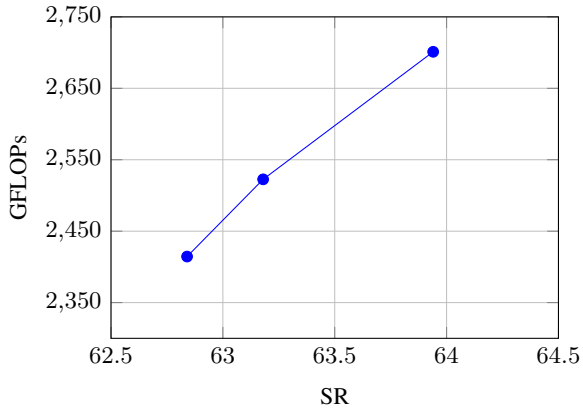

Figure 11. **Trade-off between Performance (SR) and GFLOPs.**

I. Related Work on Model Compression

Research has proposed an orthogonal approach to reduce the computational demands and memory footprint of deep-learning models: *model compression*. Quantization and pruning are the leading practice in model compression. Quantization [4, 6, 12, 13, 32, 38, 40, 44, 54] transforms the memory representation of model parameters from 32-bit floating point numbers to a lower-bit integers (e.g., 4-bit integers), thereby making it more storage efficient and lowering memory usage. Pruning [17, 18, 23, 24, 28, 42, 45] aims to create sparse models by removing parameters that are less important for maintaining performance, effectively reducing model size and computation.

While quantization and pruning have been demonstrated in simpler unimodal encoder settings for image and text, they are much more challenging in vision-language model(VLM) settings [50, 55] and largely unexplored in VLN. [55] highlighted the challenges of pruning VLMs due to the unequal weighting of visual and linguistic modalities. They mitigated this by using a modal-adaptive approach, adjusting pruning ratios across different model components based on downstream task sensitivity. Similarly, [50] demonstrated that naively applying post-training quantization to CLIP caused significant performance degradation, which they addressed by introducing prompt tuning and alignment modules.

We expect similar challenges to be exhibited by VLN agents, if not exacerbated. VLN models, in addition to processing language and visual modalities, involve sequential decision-making dependent on actions taken at each time

step. We anticipate the complex interactions between these information sources to require careful consideration while adapting model compression techniques. Future research on such techniques can be superposed along with our input-adaptive inference method to develop highly efficient models with an acceptable performance trade-off.

J. Generalizability to Other EAI Settings

Here, we discuss the applicability of our proposed techniques to additional embodied AI (EAI) settings.

Physical-world deployment. The ultimate goal of VLN research is the effective and efficient deployment of agents in the physical world. We believe our computational efficiency generalizes to real-world deployment, as physical embodied agents typically comprise building a harness around agents trained in discrete environments [2, 59]. Several challenges in this process include waypoint prediction, building navigation graphs, the visual domain gap, and latency. We address these in our work. We study the first two in our continuous environment experiment (Sec 4.2) and the visual domain gap with natural visual degradations in Sec 4.4. Our work offers a direct mechanism to address latency, which can lower barriers to practical real-world deployment.

General embodied settings. While our approach is designed for panoramic observations, it generalizes to other EAI settings. Panoramas are extensively used in non-VLN tasks, e.g., visual navigation [60], humanoid robots [64], and autonomous driving [68]. We expect high transferability to any setting employing panoramas. Generally, panoramic observations provide a wider scene context that can be valuable for decision making, albeit at the cost of computations. Our method alleviates this limitation and can facilitate wider use of panoramas for embodied AI.