Finding Visual Task Vectors

Alberto Hojel [1] **Yutong Bai** [1] **Trevor Darrell** [1] **Amir Globerson** [2 3] **Amir Bar** [1 2]

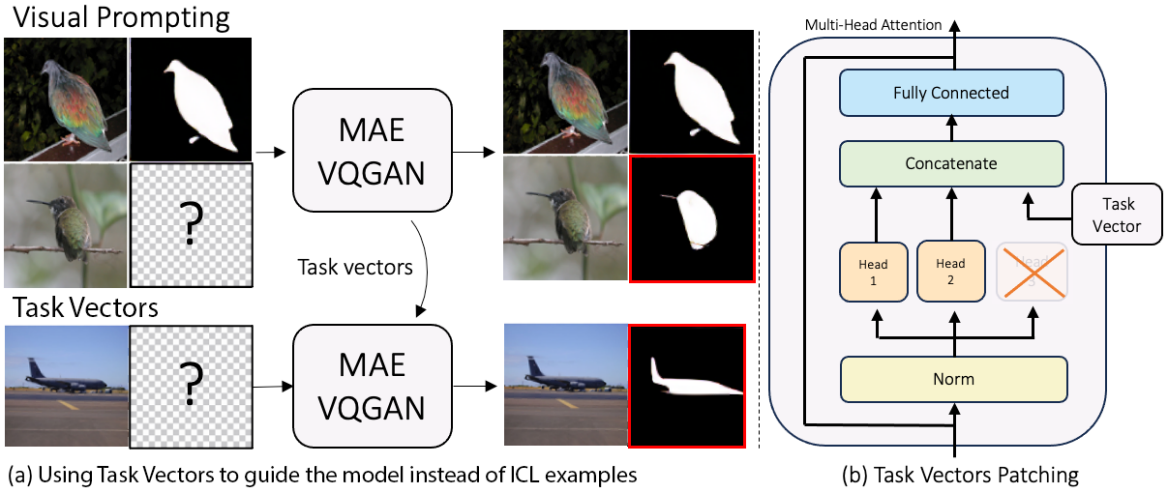

Figure 1. Visual Prompting models like MAE-VQGAN (Bar et al., 2022) require input-output example(s) to describe the desired task in their forward pass. We analyze the model activations and find *Task Vectors*, activations that encode task information that can be reused to control the task the model performs (see Figure 1a). Specifically, we tap into activations of individual attention heads and replace their outputs with Task Vectors to guide the model to the desired task (see Figure 1b). Surprisingly, the resulting models perform better than the original model while removing the need for input-output examples. This confirms that *Task Vectors* exist in the network activation space and they can guide the model to perform the desired task.

Abstract

Visual Prompting is a technique for teaching models to perform a visual task via in-context examples, without any additional training. In this work, we analyze the activations of MAE-VQGAN, a recent Visual Prompting model (Bar et al., 2022), and find *Task Vectors*, activations that encode task-specific information. We then demonstrate that it is possible to identify the Task Vectors and use them to guide the network towards performing different tasks without having to provide any in-context input-output examples. To find Task Vectors, we compute the mean activations of the attention heads in the model per task and use the REINFORCE (Williams, 1992) algorithm to patch into a subset of them with a new query image. The resulting Task Vectors guide the model with better performance than the original model.[1]

[1]UC Berkeley [2]Tel Aviv University [3]Google Research. Correspondence to: Alberto Hojel <ahojel@berkeley.edu>.

[1]For code and models see www.github.com/alhojel/visual_task_vectors

1. Introduction

In-context learning (ICL) is an emergent capability of large neural networks, first discovered in GPT-3 (Brown et al., 2020). ICL allows models to adapt to novel downstream tasks specified in the user's prompt. In computer vision, Visual ICL (known as Visual Prompting) is still at its infancy but it is increasingly becoming more popular (Bar et al., 2022; Bahng et al., 2022; Bai et al., 2023; Zhang et al., 2024b), mainly due to the appeal of using a single model to perform various downstream tasks without specific finetuning or change in the model weights.

In this work, we ask how does in-context learning work in computer vision. While this question is yet to be explored, there has been a significant body of research in Natural Language Processing (NLP) trying to explain this phenomenon (Akyürek et al., 2022; Dai et al., 2022; Garg et al., 2022; Hahn & Goyal, 2023; Han et al., 2023). Most recently, Hendel et al.(Hendel et al., 2023) suggested that LLMs encode *Task Vectors*, these are vectors that can be patched into the network activation space and replace the ICL examples while resulting in a similar functionality. Concurrently, Todd et al. (Todd et al., 2023) discovered *Function Vectors*, activations of transformer attention heads that carry

task representations. Our work is inspired by these observations and aims to study how ICL works in computer vision. We provide a more extended overview of the related works in the Suppl. Section 7.

We hypothesize that Visual ICL models create task vectors too, and aim to identify them. However, finding visual task vectors by relying on previous approaches is challenging. For example, both (Hendel et al., 2023; Todd et al., 2023) restricted their search space to the output activations of the last token in the prompt sequence. However, with images (see Figure 1, "Visual Prompting" input image), it is not obvious what token activations hold task information, and architectures like MAE-VQGAN (Bar et al., 2022) do not process image tokens sequentially. This alone increases the search space significantly because multiple tokens might hold task vectors.

To build intuition regarding the existence of visual task vectors, we visualize intermediate activations ranked by their "taskness", expecting task vectors to remain invariant within a task but vary across different tasks. Then, we propose an approach to find the subset of task vectors using REINFORCE, within the mean task activations of self-attention heads. Patching these identified task vectors leads to competitive performance with the original model across various tasks, confirming them as true task vectors. Our contributions include a method to select visual task vectors and demonstrating that these vectors enhance model performance while reducing the number of FLOPS by 22.5

2. Methods

Our goal is to understand in-context learning for computer vision and how existing models can be adapted to different downstream tasks at inference time. We focus on the MAE-VQGAN model (Bar et al., 2022), a variant of MAE (He et al., 2021) with a Vision Transformer (Dosovitskiy et al., 2021) encoder-decoder architecture. Given an input-output example (x_s, y_s) and a new query x_q, to succeed in an ICL task, the model F must implicitly apply the transformation from x_s to y_s over x_q to produce y_q:

$$y_q = F(x_s, y_s, x_q) \tag{1}$$

Based on observations from NLP (Hendel et al., 2023; Todd et al., 2023), we hypothesize that computer vision models also encode latent Task Vectors in their activation space during the forward pass. This requires the model to implicitly map the ICL example into a set of latent Task Vectors $z = \{z_i\}_{i=1}^{k}$ which we derive from the attention head outputs for different tokens across the model's layers (the internal representations). The original function F can then be decomposed into extracting the Task Vectors by a function G and applying F on the query while fixing the computed task activations $z \in \mathbb{R}^d$, where d is the hidden dimension of the model.

$$z = G(x_s, y_s) \qquad y_q = F(x_q|z) \tag{2}$$

We now describe how Task Vectors can be derived from the internal representations of a model.

2.1. Computing Mean Activations

Let $i = (l, m, k)$ denote the position in the model, where l, m, and k are the attention block, head, and token indices respectively. Define $H_i : (x_s, y_s, x_q) \rightarrow \mathbb{R}^d$ as the function that outputs the activation at position i for a given demonstration and query triplet. Let $(x_s, y_s, x_q) \sim \mathbf{D_{task_j}}$ be a triplet of input-output example and a query from task j. We compute the intermediate activation $h^i_{task_j}$ via H_i and denote its mean activation as $\mu_{i,j}$:

$$h^i_{task_j} = H_i(x_s, y_s, x_q) \qquad \mu_{i,j} = \mathbb{E}[h^i_{task_j}] \tag{3}$$

2.2. Scoring Activations

Intuitively, every task vector is an activation that changes across different tasks but remains relatively invariant to changes within a specific task. We define the data distribution $(x'_s, y'_s, x'_q) \sim \mathbf{D_{all\_tasks}}$ as the union of all task-specific distributions, and the intermediate activations $h^i_{all} = H_i(x'_s, y'_s, x'_q)$. The scoring function $\rho_{token}(i)$ is defined as the ratio of inter-task to intra-task variance as $\rho_{token}(i)$ where Var$(\cdot)$ denotes the variance, $h[e]$ is the e-th element of vector h, and n is the number of tasks.

$$\rho_{token}(i) = \frac{\sum_{e=1}^{d} \text{Var}(h^i_{all}[e])}{\frac{1}{n}\sum_{j=1}^{n}\sum_{e=1}^{d} \text{Var}(h^i_{task_j}[e])} \tag{4}$$

Computing these is efficient, requiring only forward passes across batches of data. This scoring function identifies "taskness" in activations (see Figure 2, and Table 2), but may also highlight other task-variant activations (e.g., color histograms varying across tasks but consistent within tasks like binary segmentation maps versus colored images). We use these scores mainly to build intuition, analyze task clusters in the activation space (Section 9), and develop a simple baseline ("Greedy Random Search", see Suppl. 13.1). However, we employ a more robust general approach to identifying high-order Visual Task Vectors, as described in the following section.

2.3. Finding Visual Task Vectors via REINFORCE

How can Task Vectors be identified? An exhaustive search over all subsets of activations is intractable. For example, MAE-VQGAN (Bar et al., 2022) which we utilize here has 32 attention blocks, each with 16 heads, therefore, if we

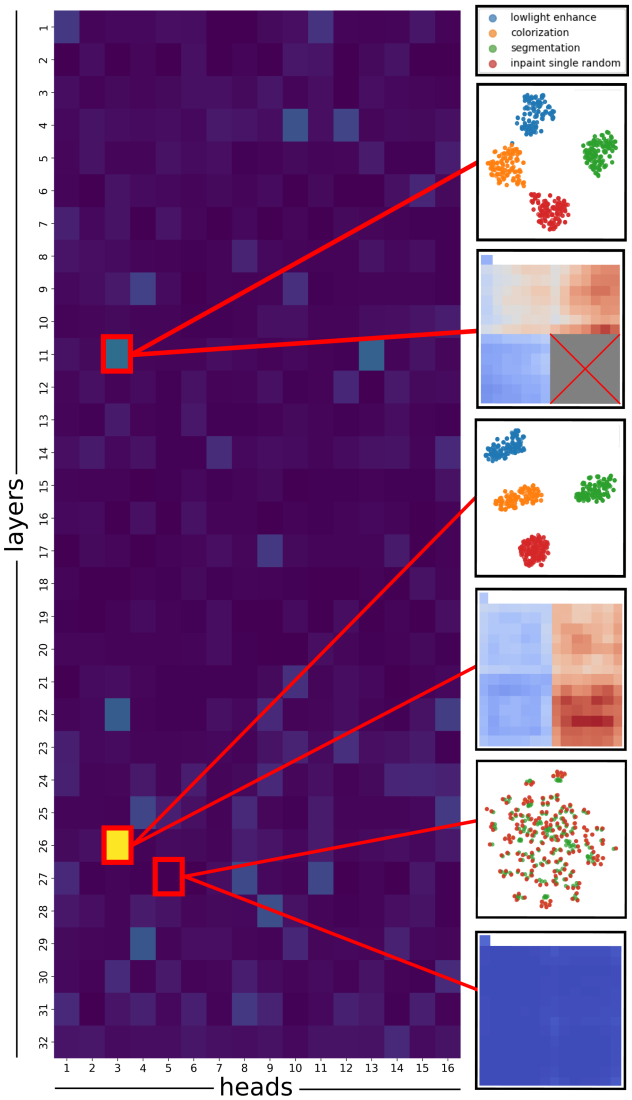

Figure 2. **Activation Scoring Analysis.** Individual scores ($\rho_{token}(i)$) aggregated per Attention Head (left). Individual token scores of specific heads are further visualized in their corresponding spatial arrangements, along with the t-SNE(van der Maaten & Hinton, 2008) clustering of head activations of different tasks (right).

consider these to be the potential set of activations then we need to search over 2^{512} options, evaluating each option over a held-out validation set.

For every task j we can apply the following procedure to find the Task Vectors. Recall that $\{\mu_{i,j}\}$ denotes the mean activations and $F(\cdot)$ is a pretrained visual prompting model. Denote $\alpha_{i,j} \sim Bernoulli(\sigma(\theta_{ij}))$ as random variables that signify whether the mean task activation $\mu_{i,j}$ is a task vector of task j placed in activation position i and θ_{ij} is a learned weight followed by the sigmoid function to ensure it is in $[0, 1]$.

Denote z_j as the set of Task Vectors for task j: $z_j = \{\mu_{i,j} | \alpha_{i,j} = 1\}$. Given pairs of input-output task demonstrations $x_q, y_q \sim \mathbf{D_{task_j}}$,[2] we want to find the set of Task Vectors that minimizes the loss function:

$$L(\theta) = E_{z_j \sim p_\theta} L(y_q, F(x_q | z_j)) \tag{5}$$

Where p_θ denotes the sampling distribution of z_j and L denotes the loss function that suits the particular task j. Note that differently than in Equation 1, if we find a good set of Task Vectors z_j, then we no longer need to condition F over additional input-output examples.

To find a set of Task Vectors, we need to estimate the parameters θ. Since the sampling distribution p_θ depends on θ, it is natural to use the REINFORCE algorithm (Williams, 1992). The key idea in REINFORCE is the observation that:

$$\nabla L(\theta) = E_{z_j \sim p_\theta} L(y_q, F(x_q | z_j)) \nabla \log p_\theta(z_j)$$

Thus, we can approximate $\nabla L(\theta)$ by sampling z_j and averaging the above equation. After iteratively optimizing with gradient descent, we select the final task vector positions by sampling a set of z_j.

Instead of learning Task Vectors placements α_{ij} for each individual task, we can revise the procedure above to learn a single placement by defining random variables $\alpha_i \sim Bernoulli(\sigma(\theta_i))$ that accommodate all tasks. In this setting, training requires drawing examples across tasks with their respective task-wise mean activations and task specific loss function. We refer to this setting as *multi-task* patching. Furthermore, the search can be applied at different granularities, such as patching token groups from the same quadrant, all tokens in an attention head, or entire layers. We discuss these design choices in the next section.

3. Experiments

We evaluate our Task Vector patching approach on four standard image-to-image tasks: Foreground Segmentation, Low Light Enhancement, In-painting, and Colorization. Using the Pascal-5i dataset (Shaban et al., 2017), we sample 1000 prompt-query pairs per split from the validation set for evaluation.

Our experiments utilize the pretrained MAE-VQGAN model from (Bar et al., 2022) as the base architecture. To find Task Vectors, we embed only the query image in the bottom left quadrant of a 2x2 grid, with the model reconstructing the output in the bottom right quadrant. We intervene on attention head outputs by replacing them with mean activations at positions identified using REINFORCE (Williams, 1992).

[2]We overload the definition of $\mathbf{D_{task_j}}$ to avoid notation clutter.

The REINFORCE algorithm is implemented as follows: We initialize Bernoulli parameters $\theta_{ij} = -1$ for each patching position. In each iteration, we sample 32 times from the Bernoulli distribution for each of 10 images, patching positions where the sampled value is 1. This results in 320 executions of task vector conditioned MAE-VQGAN per iteration. We optimize Bernoulli parameters using Adam (Kingma & Ba, 2017) with a learning rate of 0.1 for 600 steps, selecting the best checkpoint every 50 steps based on evaluation on a held-out test set.

We compare our Task Vector patching approach to several baselines: the original MAE-VQGAN one-shot performance (Bar et al., 2022), Causal Mediation Analysis (CMA) (Todd et al., 2023), Greedy Random Search (GRS), and random patching baselines. We evaluate two variants of our method: task-specific, which finds and patches Task Vectors for each task independently, and multi-task, which jointly finds Task Vectors across all tasks (patching into the same positions but using task-specific mean activations).

For evaluation, we report mean Intersection over Union (mIoU) for Segmentation and Mean Squared Error (MSE) for the other tasks, averaged across four dataset splits. To provide deeper insights into our method, we conduct ablation studies on the location of Task Vectors within the encoder and decoder, as well as the granularity of patching (individual tokens, quadrants, or attention heads).

To build intuition about the existence of Task Vectors, we analyze the clustering of activations based on our proposed scoring function $\rho_{token}(i)$. We compute this score for each attention head and visualize it as a heatmap (Figure 2, left). For selected heads, we visualize the activation clustering using t-SNE (van der Maaten & Hinton, 2008) and display the individual Activation Score per token (Figure 2, right). This qualitative analysis aims to provide visual evidence for the existence of Task Vectors and inform our Task Vector identification method.

4. Results

Our experiments demonstrate that Task Vector interventions can effectively replace in-context examples while maintaining or improving performance compared to the original MAE-VQGAN model across multiple tasks. Table 3 presents the mean and standard deviation of performance metrics across four dataset splits for various models and baselines. For a comprehensive breakdown of results across all splits and additional baselines, refer to Table 6 in the Supplementary Materials.

Our task-specific models consistently outperform the original MAE-VQGAN one-shot prompting across all evaluated tasks, despite not using input-output examples. The multi-task variant of our method also shows competitive results,

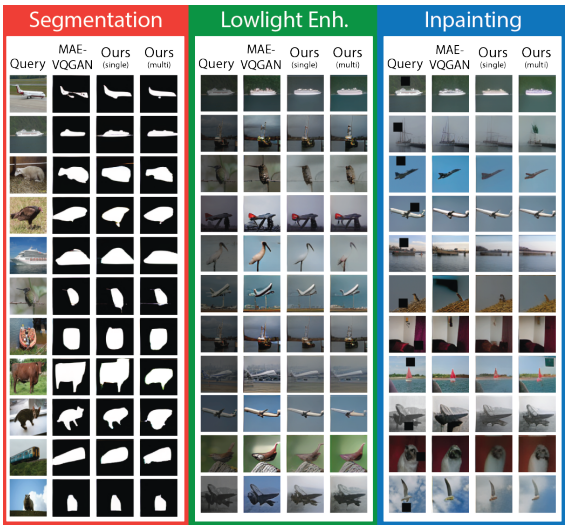

Figure 3. **Qualitative Examples.** We compare our task-specific and multitask methods to MAE-VQGAN

outperforming or matching MAE-VQGAN in all tasks except Segmentation, where it achieves comparable performance. This demonstrates the robustness of our approach in identifying task-agnostic Visual Task Vector positions that can guide the model without in-context examples.

Comparing our method to other baselines, we observe that while Causal Mediation Analysis (CMA) improves upon random baselines, it falls short of our approach in both task-specific and multi-task settings. The performance of random patching strategies (Random Quadrants and Top Quadrants) is significantly worse than other methods, highlighting the importance of our Task Vector identification process.

The qualitative analysis of attention head activations (Figure 2) provides insights into the existence and distribution of Task Vectors. Highly-ranked heads, such as (26,3), show clear task-based clustering in the t-SNE visualization, while low-ranked heads like (27,5) exhibit mixed clusters across tasks. This supports our hypothesis that certain attention heads capture task-specific information more effectively than others.

The heatmap of individual token scores within attention heads reveals interesting patterns. We observe consistency within quadrants, supporting our quadrant-based token patching strategy. The CLS token and specific quadrants show higher scores, indicating their importance in capturing task-relevant information.

Figure 3 provides qualitative examples comparing our task-specific and multi-task methods to the original MAE-VQGAN model. These visual results corroborate the quantitative improvements, showcasing how Task Vectors can guide the model to perform various tasks without relying on input-output examples.

Table 1. **Quantitative Analysis.** Results comparison across different tasks and splits, indicating the effectiveness of our task specific model. For full results across splits including more baselines, see Suppl. Table 6.

Model	Segmentation ↑ (Mean ± STD)	Lowlight Enhance ↓ (Mean ± STD)	Colorization ↓ (Mean ± STD)	In-painting ↓ (Mean ± STD)
Original MAE-VQGAN	0.338 ± 0.033	0.685 ± 0.032	0.618 ± 0.027	0.550 ± 0.042
Random Quadrants	0.170 ± 0.061	3.000 ± 0.967	3.025 ± 1.190	2.350 ± 0.955
Top Quadrants	0.150 ± 0.023	4.875 ± 0.228	4.250 ± 0.269	3.900 ± 0.141
CMA (Task-specific)	0.230 ± 0.012	0.825 ± 0.043	0.895 ± 0.063	1.750 ± 0.112
CMA (Multitask)	0.150 ± 0.017	1.400 ± 0.122	1.130 ± 0.075	1.225 ± 0.083
Ours (Multitask)	0.325 ± 0.026	0.492 ± 0.025	0.502 ± 0.036	0.558 ± 0.022
Ours (Task-specific)	$\mathbf{0.353} \pm 0.028$	$\mathbf{0.458} \pm 0.032$	$\mathbf{0.453} \pm 0.036$	$\mathbf{0.480} \pm 0.022$

For a more detailed analysis of our results, including ablation studies on the importance of layer ranking and the impact of different patching strategies, please refer to Section 6 in the Supplementary Materials.

5. Limitations

While we focused on Task Vectors, other important vector-types might exist, for example, vectors capturing image structure and ordering. We evaluated performance using MSE (except mIoU for Segmentation) after decoding VQGAN tokens. Future work could explore direct evaluation in VQGAN token space using cross entropy loss for potentially more accurate results.

6. Conclusion

In this work we explore the internal mechanisms of visual in-context learning and devise an algorithm to identify Task Vectors, activations present in transformers that can replace the in-context examples to guide the model into performing a specific task. We confirm our approach by adapting MAE-VQGAN to perform tasks without including the ICL demonstration in the prompt by patching the Task Vectors identified. We find that different than in NLP, in computer vision Task Vectors are distributed throughout the network's encoder and decoder.

Acknowledgements: This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant ERC HOLI 819080). Prof. Darrell's group was supported in part by DoD including DARPA's LwLL and/or SemaFor programs, as well as BAIR's industrial alliance programs. This work was completed in partial fulfillment for the Ph.D degree of the last author.

References

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Bahng, H., Jahanian, A., Sankaranarayanan, S., and Isola, P. Exploring visual prompts for adapting large-scale models. *arXiv preprint arXiv:2203.17274*, 2022.

Bai, Y., Geng, X., Mangalam, K., Bar, A., Yuille, A., Darrell, T., Malik, J., and Efros, A. A. Sequential modeling enables scalable learning for large vision models. *arXiv preprint arXiv:2312.00785*, 2023.

Bar, A., Gandelsman, Y., Darrell, T., Globerson, A., and Efros, A. Visual prompting via image inpainting. *Advances in Neural Information Processing Systems*, 35: 25005–25017, 2022.

Bau, D., Zhu, J.-Y., Strobelt, H., Zhou, B., Tenenbaum, J. B., Freeman, W. T., and Torralba, A. Gan dissection: Visualizing and understanding generative adversarial networks. *arXiv preprint arXiv:1811.10597*, 2018.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Dai, D., Sun, Y., Dong, L., Hao, Y., Sui, Z., and Wei, F. Why can gpt learn in-context? language models secretly perform gradient descent as meta optimizers. *arXiv preprint arXiv:2212.10559*, 2022.

Davies, D. and Bouldin, D. A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-1:224 – 227, 05 1979. doi: 10.1109/TPAMI.1979.4766909.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12873–12883, June 2021.

Ferry, Q. R., Ching, J., and Kawai, T. Emergence and function of abstract representations in self-supervised transformers. *arXiv preprint arXiv:2312.05361*, 2023.

Gandelsman, Y., Efros, A. A., and Steinhardt, J. Interpreting clip's image representation via text-based decomposition. *arXiv preprint arXiv:2310.05916*, 2023.

Garg, S., Tsipras, D., Liang, P. S., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

Hahn, M. and Goyal, N. A theory of emergent in-context learning as implicit structure induction. *arXiv preprint arXiv:2303.07971*, 2023.

Han, C., Wang, Z., Zhao, H., and Ji, H. In-context learning of large language models explained as kernel regression. *arXiv preprint arXiv:2305.12766*, 2023.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. B. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021. URL https://arxiv.org/abs/2111.06377.

Hendel, R., Geva, M., and Globerson, A. In-context learning creates task vectors. *arXiv preprint arXiv:2310.15916*, 2023.

Jia, M., Tang, L., Chen, B.-C., Cardie, C., Belongie, S., Hariharan, B., and Lim, S.-N. Visual prompt tuning. In *European Conference on Computer Vision*, pp. 709–727. Springer, 2022.

Jin, Z., Cao, P., Yuan, H., Chen, Y., Xu, J., Li, H., Jiang, X., Liu, K., and Zhao, J. Cutting off the head ends the conflict: A mechanism for interpreting and mitigating knowledge conflicts in language models. *arXiv preprint arXiv:2402.18154*, 2024.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Liu, J., Shen, D., Zhang, Y., Dolan, B., Carin, L., and Chen, W. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

Liu, S., Xing, L., and Zou, J. In-context vectors: Making in context learning more effective and controllable through latent space steering. *arXiv preprint arXiv:2311.06668*, 2023.

Lu, S., Schuff, H., and Gurevych, I. How are prompts different in terms of sensitivity? *arXiv preprint arXiv:2311.07230*, 2023.

Lu, Y., Bartolo, M., Moore, A., Riedel, S., and Stenetorp, P. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.

Luo, H. and Specia, L. From understanding to utilization: A survey on explainability for large language models. *arXiv preprint arXiv:2401.12874*, 2024.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

Moraffah, R., Karami, M., Guo, R., Raglin, A., and Liu, H. Causal interpretability for machine learning-problems, methods and evaluation. *ACM SIGKDD Explorations Newsletter*, 22(1):18–33, 2020.

Palit, V., Pandey, R., Arora, A., and Liang, P. P. Towards vision-language mechanistic interpretability: A causal tracing tool for blip. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2856–2861, 2023.

Park, K., Choe, Y. J., and Veitch, V. The linear representation hypothesis and the geometry of large language models. *arXiv preprint arXiv:2311.03658*, 2023.

Pearl, J. Direct and indirect effects. In *Probabilistic and causal inference: the works of Judea Pearl*, pp. 373–392. 2022.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rousseeuw, P. J. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL https://www.sciencedirect.com/science/article/pii/0377042787901257.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. Imagenet large scale visual recognition challenge, 2015.

Shaban, A., Bansal, S., Liu, Z., Essa, I., and Boots, B. One-shot learning for semantic segmentation. *arXiv preprint arXiv:1709.03410*, 2017.

Singh, C., Inala, J. P., Galley, M., Caruana, R., and Gao, J. Rethinking interpretability in the era of large language models. *arXiv preprint arXiv:2402.01761*, 2024.

Todd, E., Li, M. L., Sharma, A. S., Mueller, A., Wallace, B. C., and Bau, D. Function vectors in large language models. *arXiv preprint arXiv:2310.15213*, 2023.

van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Wang, B. and Komatsuzaki, A. Gpt-j-6b: A 6 billion parameter autoregressive language model, 2021.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Wu, X. and Varshney, L. R. Transformer-based causal language models perform clustering. *arXiv preprint arXiv:2402.12151*, 2024.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

Xu, J., Gandelsman, Y., Bar, A., Yang, J., Gao, J., Darrell, T., and Wang, X. Improv: Inpainting-based multimodal prompting for computer vision tasks. *arXiv preprint arXiv:2312.01771*, 2023.

Xu, S., Dong, W., Guo, Z., Wu, X., and Xiong, D. Exploring multilingual human value concepts in large language models: Is value alignment consistent, transferable and controllable across languages? *arXiv preprint arXiv:2402.18120*, 2024.

Zhang, F. and Nanda, N. Towards best practices of activation patching in language models: Metrics and methods. *arXiv preprint arXiv:2309.16042*, 2023.

Zhang, K., Lv, A., Chen, Y., Ha, H., Xu, T., and Yan, R. Batch-icl: Effective, efficient, and order-agnostic in-context learning. *arXiv preprint arXiv:2401.06469*, 2024a.

Zhang, Y., Tiňo, P., Leonardis, A., and Tang, K. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5): 726–742, 2021.

Zhang, Y., Zhou, K., and Liu, Z. What makes good examples for visual in-context learning? *Advances in Neural Information Processing Systems*, 36, 2024b.

Supplementary Material

7. Related Work

7.1. Visual Prompting

Visual Prompting (Bar et al., 2022; Bahng et al., 2022; Jia et al., 2022; Xu et al., 2023; Zhang et al., 2024b; Bai et al., 2023) is a class of approaches to adapt computer vision models to downstream tasks, inspired by the success of prompting in NLP (Brown et al., 2020). Approaches like (Bahng et al., 2022; Jia et al., 2022) seek to improve task-specific performance by adding trainable prompt vectors to the model. Other Visual Prompting approaches allow a model to handle various vision tasks (Bar et al., 2022; Xu et al., 2023; Bai et al., 2023; Zhang et al., 2024b) by introducing visual examples or text at the time of inference. Such prompting is related to the way in-context learning (Xie et al., 2021; Wei et al., 2022; Liu et al., 2021; Lu et al., 2021) operates in language models (Radford et al., 2019; Brown et al., 2020; Wang & Komatsuzaki, 2021). In fact, trainable prompts and in-context learning can be viewed as two complementary approaches for "describing" a task to a model (Li & Liang, 2021). Our goal here is to better understand the underlying mechanism of Visual ICL, and we analyze the MAE-VQGAN model presented in (Bar et al., 2022).

7.2. Explainability

Causal Interventions (Bau et al., 2018; Park et al., 2023; Pearl, 2022; Meng et al., 2022) and Activation Patching (Zhang & Nanda, 2023) are valuable tools for understanding complex neural networks' internal mechanisms, enhancing model interpretability (Zhang et al., 2021; Moraffah et al., 2020; Singh et al., 2024). These methods enable systematic examination of how models encode information and represent high-level concepts (Zhang et al., 2024a; Wu & Varshney, 2024; Lu et al., 2023). By manipulating internal states or inputs and observing output changes, they reveal causal structures and effects driving model predictions (Gandelsman et al., 2023). In this work, we employ Activation Patching (Zhang & Nanda, 2023) to improve Visual Prompting models' guidance for various computer vision tasks through targeted interventions.

7.3. Task Vectors

In (Hendel et al., 2023; Todd et al., 2023; Ferry et al., 2023), Task Vectors or Function Vectors are sets of latent activations derived from particular positions inside a transformer (Vaswani et al., 2017) model which serve as internal representations of a task implicitly described by an ICL prompt with input-output demonstrations. These latent activations at particular positions can consequently be used in a new forward pass in the absence of the ICL prompt (or with a corrupted prompt) while still managing to guide the model to perform the desired task. The investigation of Task Vectors aligns with broader efforts in the field to make neural networks more adaptable and tailored to specific tasks (Liu et al., 2023; Luo & Specia, 2024) as well as boosting the performance (Palit et al., 2023; Xu et al., 2024; Jin et al., 2024) by gaining a deeper understanding of how different attention heads within a model contribute to its overall function. Our work is the first to explore Task Vectors in computer vision.

8. Experiments

Our experiments explore if activation patching of Task Vectors can make a model perform the desired visual task as well as or better than its original one-shot performance. We describe the implementation of MAE-VQGAN, prompting schemes, baselines for comparison, visual tasks, and conducted experiments.

8.1. Implementation Details

MAE-VQGAN (Bar et al., 2022). An MAE (He et al., 2021) with a ViT-L (Dosovitskiy et al., 2021) backbone. The decoder predicts a distribution over a VQGAN (Esser et al., 2021) codebook to output images with better visual quality. We used the pretrained checkpoint from (Bar et al., 2022) where it was trained over the Computer Vision Figures (Bar et al., 2022) dataset and ImageNet (Russakovsky et al., 2015).

Finding Task Vectors. Similar to one-shot (Section 8.4), we use a 2x2 image grid with the prompt. For task vectors, we embed only the query in the bottom left quadrant. The model reconstructs only the output part (bottom right). We patchify the query image at 112x112 resolution, apply bottom left quadrant positional encodings, feed patches into the encoder, and process them with the decoder alongside bottom right quadrant mask tokens to obtain the result.

We intervene on attention head outputs by replacing them with mean activations at positions identified using REINFORCE (Williams, 1992). Initially, we set $\theta_{ij} = -1$. To reduce the search space, we group patching positions into three categories: CLS (1 token), bottom left quadrant (49 query image patch tokens), and bottom right quadrant (49 MASK tokens). In each iteration, we sample 32 times from the Bernoulli distribution for each of 10 images, patching positions where the sampled value is 1. This results in 320 executions of task vector conditioned MAE-VQGAN per iteration. We optimize Bernoulli parameters as outlined in Section 2.3 using Adam (Kingma & Ba, 2017) with a learning rate of 0.1. The algorithm runs for 600 steps, selecting the best checkpoint every 50 steps based on evaluation on a held-out test set.

8.2. Activation Scoring Analysis

Here our goal is to evaluate the Activation Scoring step (outlined in Section 2.2), specifically whether high scoring activations indeed correspond to Task Vectors.

Collecting Activations. To compute activation scores, we first run the model's forward pass in a one-shot setting across different tasks (Section 2.1). We use 100 prompts and queries from Pascal 5i (Shaban et al., 2017) training set, ensuring reasonable one-shot performance. We save the activations for each task j and position $i = (l, m, k)$, where l, m, and k are the attention block, head, and token indices respectively. We then compute the mean activation $\mu_{i,j}$ and score $\rho_{token}(i)$.

Evaluation via Clustering. Next, we wish to analyze if $\rho_{token}(i)$ indeed captures "taskness". Intuitively, we expect layers that capture task information to succeed in clustering activations by task. To assess this, we analyze the clustering performance of vectors with high-ranking activations versus those marked with low scores. We measure the clustering performance using common clustering metrics like the Silhouette Score (Rousseeuw, 1987) and the Davies-Bouldin Score (Davies & Bouldin, 1979). Finally, we also perform a qualitative analysis by visualizing the representations on a t-SNE (van der Maaten & Hinton, 2008) plot, coloring each data point by it's task label.

8.3. Downstream Tasks

We evaluate the performance on standard image-to-image tasks like Foreground Segmentation, Low Light Enhancement, In-painting, and Colorization.

Dataset. We utilize Pascal-5i (Shaban et al., 2017), consisting of 4 image splits (346-725 images each) with segmentation masks. For evaluation, We sample 1000 prompt-query pairs per split from the validation set. For Activation Scoring and methods to find Task Vectors, we use only training set examples.

Foreground Segmentation. We use Pascal-5i (Shaban et al., 2017) segmentation masks and report mean IOU (mIOU) across four splits.

Low Light Enhancement. We multiply Pascal-5i image color channels by 0.5 for input, using the original as output. We report Mean Squared Error (MSE).

Inpainting. We mask a random 25% square region (1/8 area) of each image for input. We report MSE using the original image as output.

Colorization. To obtain input-output pairs, we convert an image to grayscale and denote it as the input, and have the output be the original image. For evaluation, we report the MSE metric.

8.4. Baselines

One-shot Prompting. We follow the basic one-shot setup in (Bar et al., 2022). Specifically, we construct a grid-like image structure with an input-output demonstration, a query, and a masked output region which are embedded into a $2x2$ grid. We feed this grid image to the model to obtain the output prediction which we use for evaluation purposes.

Causal Mediation Analysis. We compare our methodology with the Causal Mediation Analysis methodology as presented in (Todd et al., 2023) as a baseline. We select the top 25% of activations with the highest causal score across 10 images.

Greedy Random Search. We compare our methodology to an iterative greedy random search algorithm (GRS) used to select Task Vectors based on the activation scoring metric proposed in Section 2.2. This serves as a baseline and is outlined in the Supplementary Materials (Section 13.1).

8.5. Ablations

In this section, we describe the set of experiments conducted to validate our implementation choices of our REINFORCE (Williams, 1992) method.

Task Vectors Location in Encoder vs. Decoder. We hypothesize that task implementation spans both encoder and decoder. To test this, we apply interventions to the encoder only, decoder only, and the whole network. We report mIoU on four Segmentation task splits to assess the necessity of interventions in both parts for effective task implementation.

Patching Granularity. We explore intervention granularities by grouping token positions to reduce dimensionality and optimization search space. Positions $i = (l, m, k)$ are grouped into spatial quadrants or attention heads. To balance precision and search space, we use three granularity levels: individual tokens, quadrants, and attention heads, and report performance across four tasks.

8.6. Task Vector Patching

We investigate whether patching Task Vectors into a model's forward pass, without ICL demonstrations, can achieve performance comparable to one-shot prompting. Our experiments include:

Task-specific. For each of the four tasks we execute our task-specific method, and the baselines of Causal Mediation Analysis (CMA) and Greedy Random Search (GRS) with 10 images and report the results on corresponding tasks.

Multi-task. We apply our method in a multi-task scenario, selecting two images per task and evaluating alongside CMA and GRS baselines. We jointly assess loss across all tasks, normalizing per task to ensure equal weighting and prevent single-task dominance. We include an identity-copy task to maintain a consistent batch size of 10 for fair comparisons.

We also provide the following baselines to benchmark the performance of the three algorithms, and to validate the necessity of top k score ranking the layers and performing the iterative search for the GRS method:

MAE-VQGAN. We compare to the original model's one-shot performance.

Random Quadrants. We patch into randomly sampled positions across the whole network (same total amount of patches as task-specific and multi-task)

GRS across Random K Layers. We execute the search algorithm on a random set of k layers, instead of score-ranked layers, to validate the necessity of our proposed scoring method.

Patching into Top Quadrants. We patch into the top quadrants based on their scoring (same total amount of patches as task-specific and multi-task). That is, we directly patch into areas with high scores naively without the iterative refining steps of the GRS, to validate its utility.

We report results across four splits for all four tasks, comparing our variants to the original MAE-VQGAN model and random baselines. Supplementary Materials (Section 8) include initial explorations of REINFORCE-powered Language Task Vectors with Llama7B, vector addition to create composable tasks, and others.

9. Results

9.1. Activation Scoring Analysis

We compute and display $\rho_{token}(i)$ per head on a heatmap, with layers on the y-axis and head indices on the x-axis (Figure 2). This highlights heads that potentially hold Task Vectors. We select top heads—$(26, 3)$ and $(11, 3)$—and a lower-ranked head $(27, 5)$. For each, we visualize activation clustering and individual Activation Score per token.

Clustering Visualization. Highly-ranked heads (e.g., (26,3)) show clear task-based clustering, while low-ranked heads like (27,5) show many small clusters with different tasks, likely based on input semantics. The activations are projected onto 2D using t-SNE (van der Maaten & Hinton, 2008) with colors indicating tasks.

Score-per-token Heatmap. We display $\rho_{token}(i)$ values for each token, reflecting spatial positioning in a 2x2 grid. For heads of interest, these values are shown on a heatmap. The CLS token is in the top left, followed by values for quadrants (x_s, y_s, x_q, y_q). The encoder lacks y_q tokens (bottom right), marked as X. Token values within a head vary widely but show consistency within quadrants, supporting quadrant-based token patching.

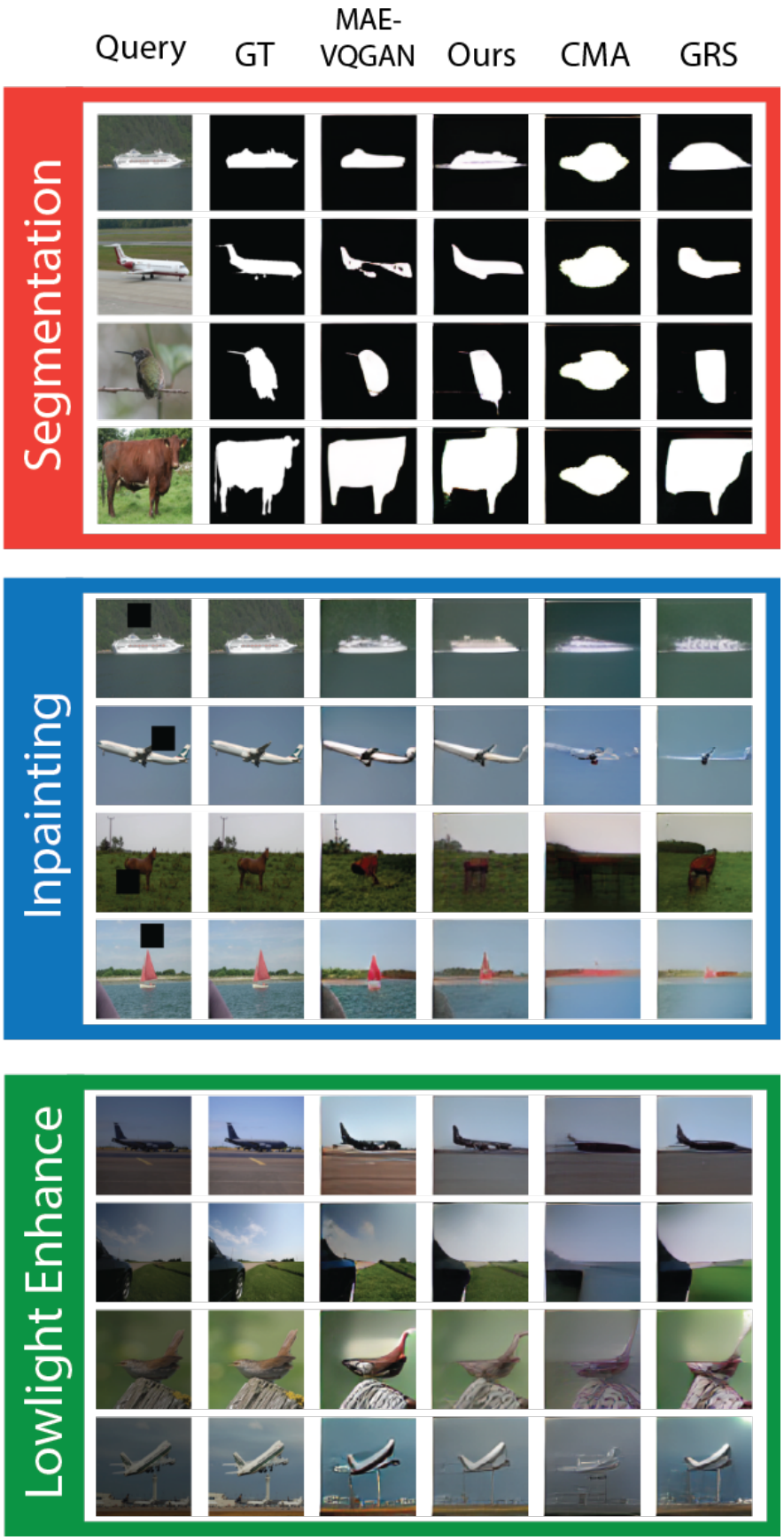

Figure 4. **Qualitative Examples.** We qualitatively compare the task-specific variants of our methodology's results with the original model and the CMA and GRS baselines. Our patching methodology performs better than the original MAE-VQGAN model.

Table 2. **Task Clustering Quality.** Clustering Scores of Different Attention Heads, ranked by our Activation Score (see Section 2). This indicates that higher Activation Scores indeed correlate with better clustering by tasks.

	(Layer, Head)	Our Score ↑	Silhouette Score ↑	Davies-Bouldin Score ↓
High 1	(26, 3)	**2.1663**	**0.3583**	**1.2744**
High 2	(11, 3)	1.0827	0.2692	1.5567
Random 1	(4, 3)	0.2329	0.0708	4.1062
Random 2	(18, 16)	0.1259	0.0369	4.3256
Low 1	(2, 16)	0.0221	-0.0518	21.8265
Low 2	(2, 12)	0.0264	-0.0334	13.5982

Table 3. **Quantitative Analysis.** Results comparison across different tasks and splits, indicating the effectiveness of our task-specific model.

Model	Segmentation ↑ (Mean ± STD)	Lowlight Enhance ↓ (Mean ± STD)	Colorization ↓ (Mean ± STD)	In-painting ↓ (Mean ± STD)
Original MAE-VQGAN	0.338 ± 0.033	0.685 ± 0.032	0.618 ± 0.027	0.550 ± 0.042
Random Quadrants	0.170 ± 0.061	3.000 ± 0.967	3.025 ± 1.190	2.350 ± 0.955
Random K Layers	0.090 ± 0.007	1.825 ± 0.043	0.568 ± 0.022	0.875 ± 0.100
Top Quadrants	0.150 ± 0.023	4.875 ± 0.228	4.250 ± 0.269	3.900 ± 0.141
CMA (Task-specific)	0.230 ± 0.012	0.825 ± 0.043	0.895 ± 0.063	1.750 ± 0.112
CMA (Multi-task)	0.150 ± 0.017	1.400 ± 0.122	1.130 ± 0.075	1.225 ± 0.083
GRS (Task-specific)	0.320 ± 0.021	0.600 ± 0.025	0.555 ± 0.025	0.580 ± 0.047
GRS (Multi-task)	0.323 ± 0.019	0.515 ± 0.032	0.568 ± 0.029	0.605 ± 0.036
Ours (Multi-task)	0.325 ± 0.026	0.492 ± 0.025	0.502 ± 0.036	0.558 ± 0.022
Ours (Task-specific)	$\mathbf{0.353} \pm 0.028$	$\mathbf{0.458} \pm 0.032$	$\mathbf{0.453} \pm 0.036$	$\mathbf{0.480} \pm 0.022$

Quantitative Clustering Analysis. To validate our scoring-based clustering quality observations, we report Silhouette and Davies-Bouldin scores for the two highest-ranked heads, two randomly sampled heads, and the two lowest-ranked heads (Table 2). Heads scored highly by our method show high-quality clustering scores, while low-scored heads show poor clustering. Randomly selected heads received intermediate scores, further supporting our methodology.

9.2. Task Vector Patching

We present the performance of our task-specific and multi-task methods compared to the original model and baselines. Task Vector interventions improve visual ICL task performance over the original model. Table 3 shows mean and variance across 4 splits (full results in Table 6, Supplementary Materials). Our task-specific models outperform MAE-VQGAN in all tasks, with GRS models surpassing it in some. Our multi-task models excel in all tasks except Segmentation, where they match MAE-VQGAN. GRS multi-task matches the original model, while CMA improves upon random baselines but falls short of our method and GRS.

We validate the importance of ranking layers by $\rho_{layer}(l)$ (Section 13.1) and selecting top-k for greedy random search performance. Random K layers generally underperform, except in colorization. Simply patching top quadrants without GRS also yields poor results. Qualitative results for Segmentation, Lowlight Enhancement, and In-painting are presented. We compare task-specific models of our method with original MAE-VQGAN, CMA, and GRS baselines (Figure 4), and visualize task-specific and multi-task variants against CMA and GRS (Figure 5).

9.3. Ablations

Task Vectors Location in Encoder vs. Decoder. We compare interventions isolated to the encoder, decoder, and throughout the whole network. Results show that in-context task learning utilizes both components, with the decoder playing a more crucial role. Intervening in both components is essential for task implementation, supporting our hypothesis of distributed computation with cascading effects throughout the network (see Table 5).

Patching Granularity. Inspired by quadrant patterns in per-token scoring, we explore optimal token grouping to reduce

Table 4. **Optimal Patching Granularity.** Patching into Tokens (T), Quadrants (Q), or Heads (H)

Model	Segmentation ↑ (Mean ± STD)	Lowlight Enhance ↓ (Mean ± STD)	Colorization ↓ (Mean ± STD)	In-painting ↓ (Mean ± STD)
Ours T (Task-specific)	**0.350** ± 0.025	0.495 ± 0.036	**0.453** ± 0.036	0.485 ± 0.018
Ours Q (Task-specific)	0.338 ± 0.023	**0.458** ± 0.032	0.465 ± 0.036	**0.480** ± 0.022
Ours H (Task-specific)	0.245 ± 0.015	0.942 ± 0.070	0.857 ± 0.069	0.885 ± 0.067
Ours T (Multi-task)	0.318 ± 0.023	0.510 ± 0.028	0.510 ± 0.039	0.565 ± 0.025
Ours Q (Multi-task)	**0.325** ± 0.026	**0.492** ± 0.025	**0.502** ± 0.036	**0.558** ± 0.022
Ours H (Multi-task)	0.253 ± 0.013	1.105 ± 0.064	0.998 ± 0.069	0.980 ± 0.082

Table 5. **Isolating Task Locations.** Patching into Encoder only, Decoder only, and both.

	Segmentation ↑			
Model	Split 0	Split 1	Split 2	Split 3
Encoder (Task-specific)	0.09	0.14	0.14	0.13
Decoder (Task-specific)	0.32	0.34	0.29	0.29
Both (Task-specific)	**0.35**	**0.35**	**0.31**	**0.29**

search space dimensionality. Quadrant grouping improves performance for Segmentation and Colorization, while token-level granularity is better for Lowlight Enhancement and In-painting (see Table 4 for mean and variance across 4 splits, and Table 7 in Supplementary Materials for individual split evaluations).

10. Full Results

As supplementary material, we provide the tables displaying the full evaluations across the 4 splits of our method alongside the baselines, and the different patching granularities.

11. Additional Explorations

We include a handful of additional experiments that serve as potential future avenues for research.

Finding Task Vectors in Other Architectures. We evaluate our method to find Task Vectors on Llama2 7B, an autoregressive decoder-only architecture on NLP tasks and compare the Top-1 accuracy to previous reported results by Todd et al. (Todd et al., 2023). Our method performs better than previous approaches, beating 10-shot in 2 tasks while reducing FLOPs by 92.5% (Table 8).

Out of Domain Evaluation. We evaluate the performance of in-painting and low-light enhancement task-specific models on x-ray images. The qualitative results (Figure 6) show proper task implementation. The resulting x-ray images are slightly more blurry, which is due to a limitation in the underlying VQGAN tokenizer, previously reported in Bar et al. (Bar et al., 2022).

Task Vector Arithmetic. We explore whether vector arithmetic can be performed on the task representations to create new tasks by composition. By utilizing the multitask patching positions, we can compose the tasks of in-painting and segmentation by combining their individual Task Vectors as follows: $combined = inpaint + segment - identity$ similarly to Todd et al. (Todd et al., 2023), where identity is the identity mapping task described in the paper. We compute these new mean activations and patch them into the previously determined multitask positions, and evaluate using as input a masked image and its corresponding segmentation. Figure 7 shows that the tasks are indeed composable, where $combined$ performs qualitatively better than $segment$ which suffers from holes.

One-shot Task Vectors. We investigate whether 1-shot performance can be improved with Task Vectors. Table 9 shows that additional in-context examples (or shots) are not necessary when using Task Vectors. Intuitively, Task Vectors already lead

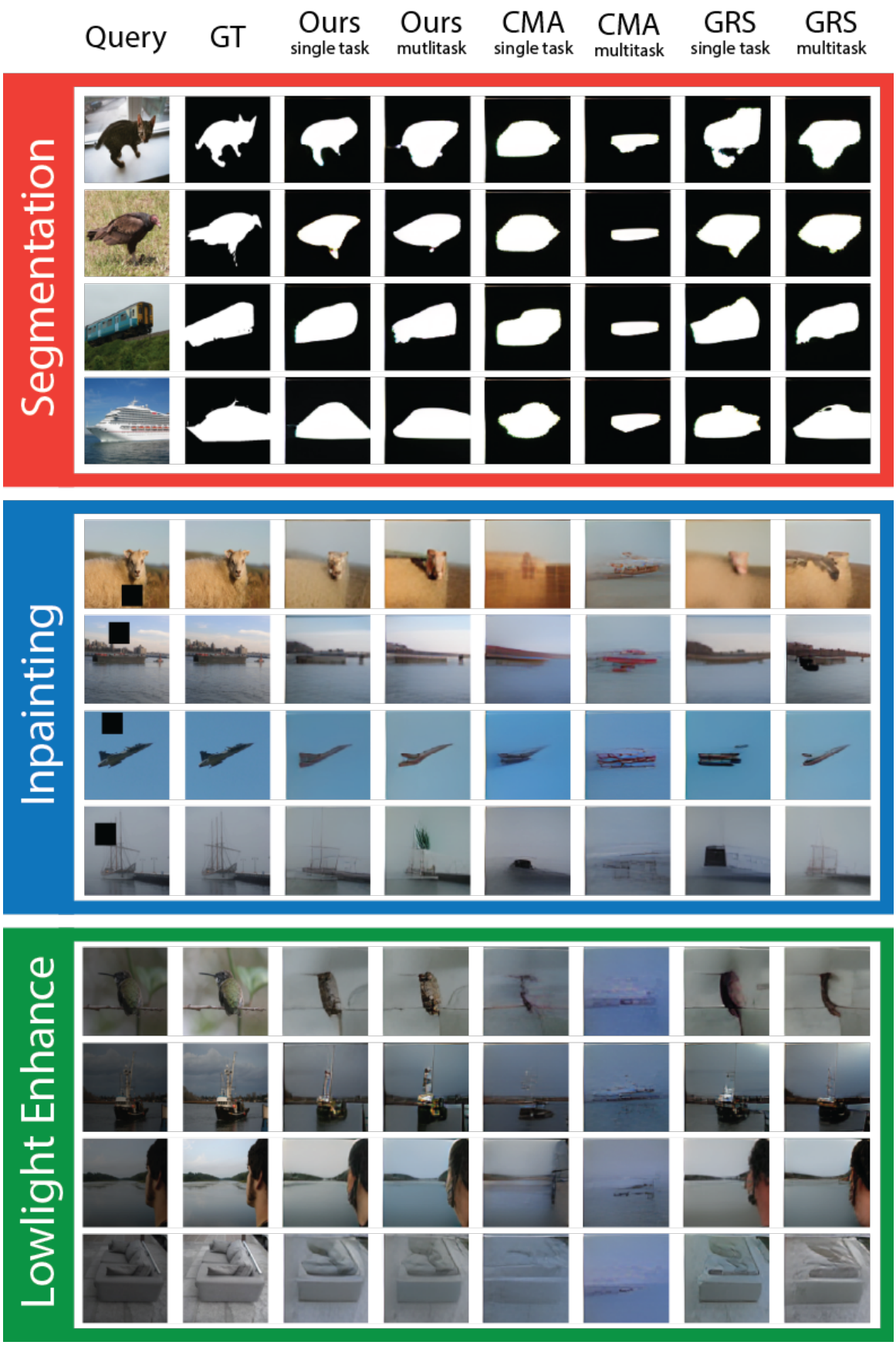

Figure 5. **Qualitative Examples.** We qualitatively compare the task-specific and multi-task variants of our methodology with the CMA and GRS baselines. Our patching methodology performs better than the original MAE-VQGAN model.

Table 6. **Quantitative Analysis.** Results comparison across different tasks and splits, indicating the effectiveness of our task-specific model.

	Segmentation ↑				Lowlight Enhancement ↓			
Model	Split 0	Split 1	Split 2	Split 3	Split 0	Split 1	Split 2	Split 3
Original MAE-VQGAN	0.35	0.38	0.33	0.29	0.70	0.66	0.73	0.65
Random Quadrants	0.08	0.25	0.16	0.19	4.30	2.40	3.50	1.80
Random K Layers	0.09	0.10	0.09	0.08	1.80	1.80	1.90	1.80
Top Quadrants	0.11	0.17	0.16	0.16	4.50	5.00	5.10	4.90
CMA (Task-specific)	0.23	0.25	0.22	0.22	0.76	0.83	0.88	0.83
CMA (Multi-task)	0.18	0.14	0.14	0.14	1.2	1.5	1.5	1.4
GRS (Task-specific)	0.33	0.35	0.30	0.30	0.56	0.61	0.63	0.60
GRS (Multi-task)	0.33	0.35	0.31	0.30	0.47	0.52	0.56	0.51
Ours (Multi-task)	0.35	0.35	0.31	0.29	0.46	0.49	0.53	0.49
Ours (Task-specific)	**0.38**	**0.38**	**0.33**	**0.32**	**0.41**	**0.46**	**0.50**	**0.46**
	Colorization ↓				In-painting ↓			
Model	Split 0	Split 1	Split 2	Split 3	Split 0	Split 1	Split 2	Split 3
Original MAE-VQGAN	0.59	0.62	0.66	0.60	0.49	0.55	0.61	0.55
Random Quadrants	2.10	4.10	4.30	1.60	3.80	1.20	1.90	2.50
Random K Layers	0.54	0.57	0.60	0.56	0.72	0.89	1.00	0.89
Top Quadrants	3.80	4.40	4.30	4.50	3.70	3.90	4.10	3.90
CMA (Task-specific)	0.79	0.92	0.96	0.91	1.6	1.8	1.9	1.7
CMA (Multi-task)	1.02	1.2	1.2	1.1	1.1	1.3	1.3	1.2
GRS (Task-specific)	0.52	0.56	0.59	0.55	0.52	0.56	0.65	0.59
GRS (Multi-task)	0.53	0.57	0.61	0.56	0.55	0.61	0.65	0.61
Ours (Multi-task)	0.45	0.51	0.55	0.50	0.53	0.56	0.59	0.55
Ours (Task-specific)	**0.40**	**0.46**	**0.50**	**0.45**	**0.45**	**0.49**	**0.51**	**0.47**

Table 7. **Optimal Patching Granularity.** Patching into Tokens (T), Quadrants (Q), or Heads (H)

	Segmentation ↑				Lowlight Enhancement ↓			
Model	Split 0	Split 1	Split 2	Split 3	Split 0	Split 1	Split 2	Split 3
T (Task-specific)	**0.38**	**0.37**	**0.33**	**0.32**	0.44	0.50	0.54	0.50
Q (Task-specific)	0.36	0.36	0.32	0.31	**0.41**	**0.46**	**0.50**	**0.46**
H (Task-specific)	0.24	0.27	0.23	0.24	0.83	0.97	1.02	0.95
T (Multi-task)	0.34	0.34	0.30	0.29	0.47	0.51	0.55	0.51
Q (Multi-task)	**0.35**	**0.35**	**0.31**	**0.29**	**0.46**	**0.49**	**0.53**	**0.49**
H (Multi-task)	0.26	0.27	0.24	0.24	1.01	1.12	1.19	1.10
	Colorization ↓				In-painting ↓			
Model	Split 0	Split 1	Split 2	Split 3	Split 0	Split 1	Split 2	Split 3
T (Task-specific)	**0.40**	**0.46**	**0.50**	**0.45**	0.46	0.49	0.51	0.48
Q (Task-specific)	0.41	0.47	0.51	0.47	**0.45**	**0.49**	**0.51**	**0.47**
H (Task-specific)	0.75	0.88	0.94	0.86	0.78	0.92	0.96	0.88
T (Multi-task)	0.45	0.51	0.56	0.52	0.53	0.57	0.60	0.56
Q (Multi-task)	**0.45**	**0.51**	**0.55**	**0.50**	**0.53**	**0.56**	**0.59**	**0.55**
H (Multi-task)	0.89	1.02	1.08	1.0	0.85	1.02	1.07	0.98

to improved performance and additional in-context learning examples do not add more information.

12. More Qualitative Examples

We provide a wider selection of examples comparing our task vector patching methodology in comparison to **1) the original one-shot MAE-VQGAN, CMA, and GRS, and 2) our ablations.** Alongside each figure, we accompany it with an according analysis.

Table 8. Comparison of Task Vector methods on Llama2 7B for various NLP tasks

Method	Landmark to Country	Present to Past	Country to Capital
0-shot	0.0	0.084	0.047
2-shot	0.868	0.95	0.92
10-shot	**0.88**	0.967	0.951
0-shot + CMA	0.691	0.88	0.825
0-shot+ Ours	0.8629	**0.983**	**0.9524**

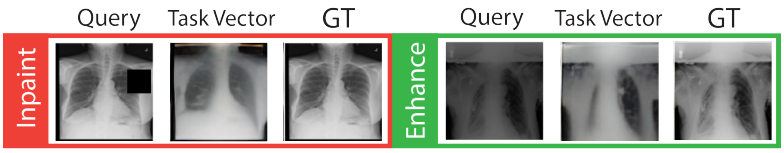

Figure 6. Out of domain evaluation on x-ray images.

First, we visualize the task-specific models of our methodology alongside the original MAE-VQGAN, and the CMA and GRS baselines (see Figure 8). Secondly, we visualize the task-specific and multi-task variants of our methodology in comparison to CMA (see Figure 9).

Qualitative Analysis for Segmentation. In Figure 10, we compare our methodology and GRS task-specific and multi-task methods to the original one-shot MAE-VQGAN performance on the task of Segmentation. It appears that our method is good at segmenting the coarse and fine details of the object of focus. In many cases, the segmentations generated by the original MAE-VQGAN suffer from holes or incomplete masks. In contrast, our method outputs consistent and coherent masks. On the other hand, the GRS method suffers with particular details especially observable when attempting to segment an animal's ear or leg. However, in many such cases it performs better than MAE-VQGAN at getting the general shape of objects.

Qualitative Analysis for Lowlight Enhancement. In Figure 11, we compare our methodology and GRS task-specific and multi-task methods to the original one-shot MAE-VQGAN performance on the task of Lowlight Enhancement. It appears that the GRS method suffers in maintaining the visual qualities of the query image. However there are many cases where MAE-VQGAN assigns bright colors which is likely due to the particular prompt in use and the inherent ambiguities of the task. On the other hand, our method–particularly the multi-task variant–ouputs consistently better results with accurate visual qualities. In some cases our method produces somewhat muted or blurry results which may be a consequence of using MSE in the pixel space as supervision, but nonetheless reports better quantitative performance.

Qualitative Analysis for In-painting. In Figure 12, we compare our methodology and GRS task-specific and multi-task methods to the original one-shot MAE-VQGAN performance on the task of In-painting. We observe that our method consistently outperforms the original model. However, it appears that the GRS task-vector patching method–once again–suffers in maintaining the higher frequency components of the query image; it appears to reduce the contrast of the image and reduce saturation. However, there are many such cases where the original MAE-VQGAN one-shot technique fails to appropriately implement the task while our method succeeds. The original model's performance depends heavily on the specific prompt used which may be the root cause of failures while task-vector patching succeeds.

Qualitative Analysis of Ablations. Finally, we present the qualitative analysis of the different ablations for the Segmentation task in Figure 13. The benefits of observing the visual features of the different ablations in addition to quantitative analysis becomes clear when comparing the Decoder Only and Encoder Only columns. Here it is clear that patching into decoder is of utmost importance in relation to patching into the encoder; the distinction is clear when observing qualitative features. In the end, it is both parts in synchrony which allow for the implementation of in-context learning.

13. Greedy Random Search baseline

13.1. Selecting Task Vectors via Greedy Random Search

For every task j we apply the following algorithm to obtain a task-specific model.

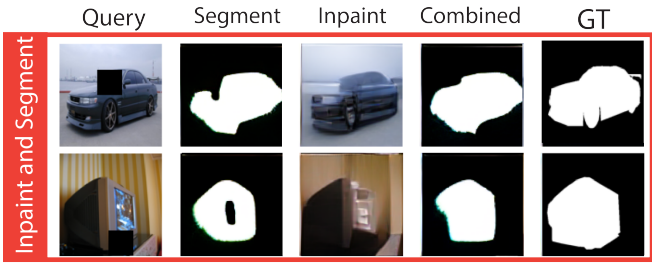

Figure 7. Task vector arithmetic results.

Table 9. Comparison of no ICL demonstration and one-shot performance with Task Vectors (TV)

Method	Segm mIoU	LowLight MSE	Inpaint MSE	Colorize MSE
Original one-shot	0.338	0.6	0.55	0.618
No ICL + TV	**0.353**	**0.458**	0.480	**0.453**
One-shot + TV	0.346	0.496	**0.454**	0.490

Input. The mean activations $\{\mu_{i,j}\}$, pretrained visual prompting model $F(\cdot)$, an evaluation set, and the aggregated per-layer score defined as $\rho_{layer}(l)$:

$$\rho_{layer}(l) = \sum_{m,k} \rho_{token}(i = (l, m, k)) \tag{6}$$

Initialization. Initialize a set of binary indicators $\{\alpha_{i,j}\}$ for all i, where $\alpha_{i,j} \in \{0, 1\}$ signifies whether the mean task activation $\mu_{i,j}$ is a task vector. Next we describe the algorithm to choose the values of $\alpha_{i,j}$.

For every activation $i = (l, m, k)$ in the top k scoring layers w.r.t $\rho_{layer}(l)$, randomly choose the value of α_i by sampling from a Bernoulli random variable with parameter value p. Set the activation vectors to be $z_j = \{\mu_{i,j} | \forall ij \text{ if } \alpha_{i,j} = 1\}$. Evaluate the now task-specific model $F(\cdot|z_j)$ on a held-out validation set. Run for T trials and for every i, j set the values of $\alpha_{i,j}$ to be the values from the most successful trial.

Greedy Search. Iterate over l in the top k scoring layers sorted by $\rho_{layer}(l)$ from high to low, pick activation $i = (l, m, k)$, flip the value $\alpha_{i,j}$ and evaluate the validation score for $F(\cdot|z_j)$. After having evaluated each flip of the $\alpha_{i,j}$ on the particular layer l we keep the $\alpha_{i,j}$ which performed best (based on a certain evaluation function) or continue to the next layer if the performance did not improve.

Termination. When one search loop across the k layers results in no changes - or after $10k$ iterations, the search has thus converged and we return the single-task model $F(\cdot|z_j)$.

This procedure is outlined for every token, attention head, and layer. However, it is possible to apply it in different levels of granularity. For example, by patching group of tokens from the same quadrant, patching all the tokens in an attention head, or patching the entire layer. We discuss these design choices in the next section.

13.2. Greedy Random Search Implementation Experiments

In this section we describe the set of experiments conducted to ascertain the particular implementation details of the greedy random search, validating the design choices.

Implementation Details We search through the top $k = 17$ layers ranked by Activation Scoring. During the initialization phase we sample $\alpha_{i,j} \in \{0, 1\}$ from a Bernoulli distribution with a parameter of $p = 0.3$ (probability of selecting 1) and evaluate performance. We repeat this for $T = 100$ trials and continue with the best performing $\alpha_{i,j}$.

Furthermore, we perform a grouping of token positions i in each individual attention head into 3 groups: the CLS token, bottom left quadrant, and bottom right quadrant. This serves to further reduce the search space. We use a set of 10 training

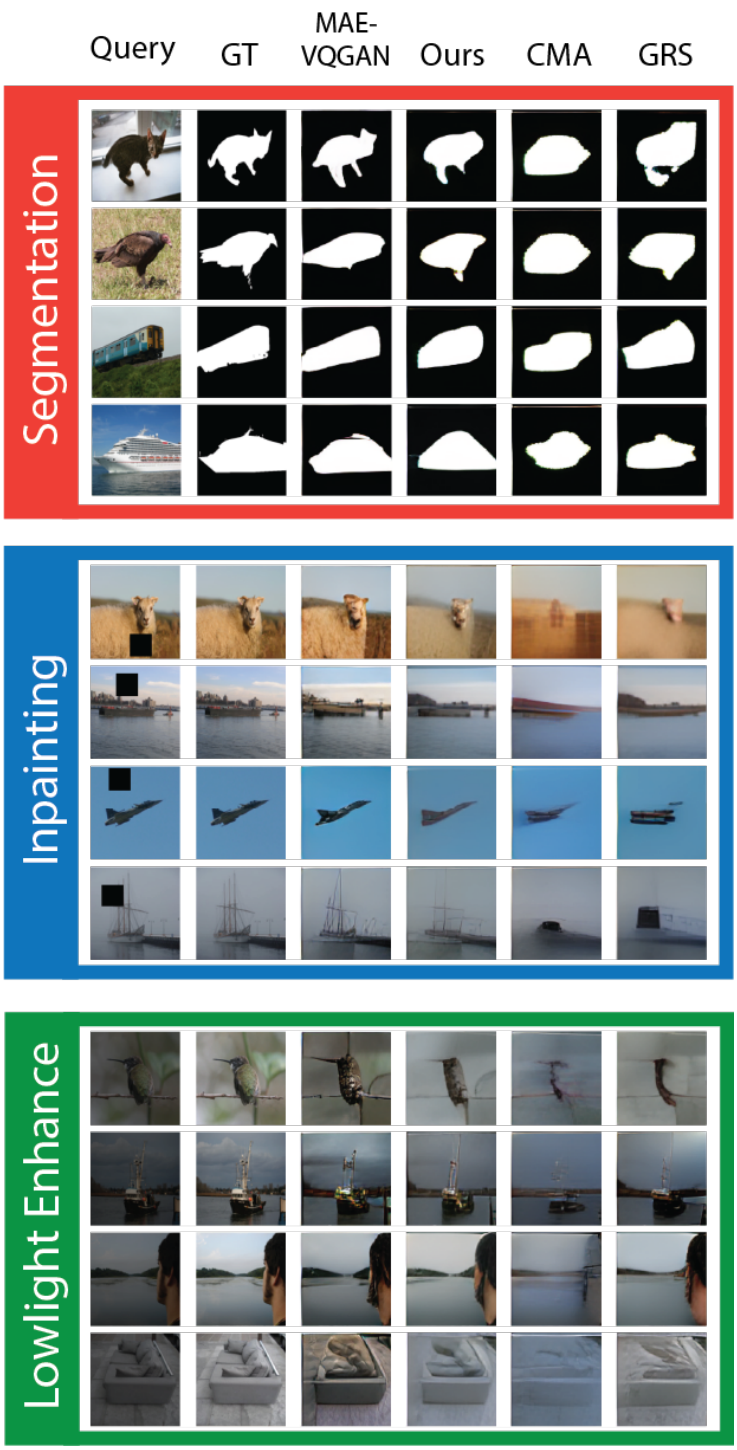

Figure 8. **Qualitative Examples.** We qualitatively compare our the task-specific variants of our methodology with the original model and the CMA and GRS baselines.

images to supervise the search. These design decisions are further validated through ablation experiments. **Selecting Initialization Parameters.** For the initialization of the Greedy Random Search there are two parameters, k which dictates how many layers to search across and the Bernoulli random variable parameter p which dictates the probability at which we set $\alpha_{i,j}$ to be 1 during the initialization phase. The question is, which k value is best at narrowing down the search space

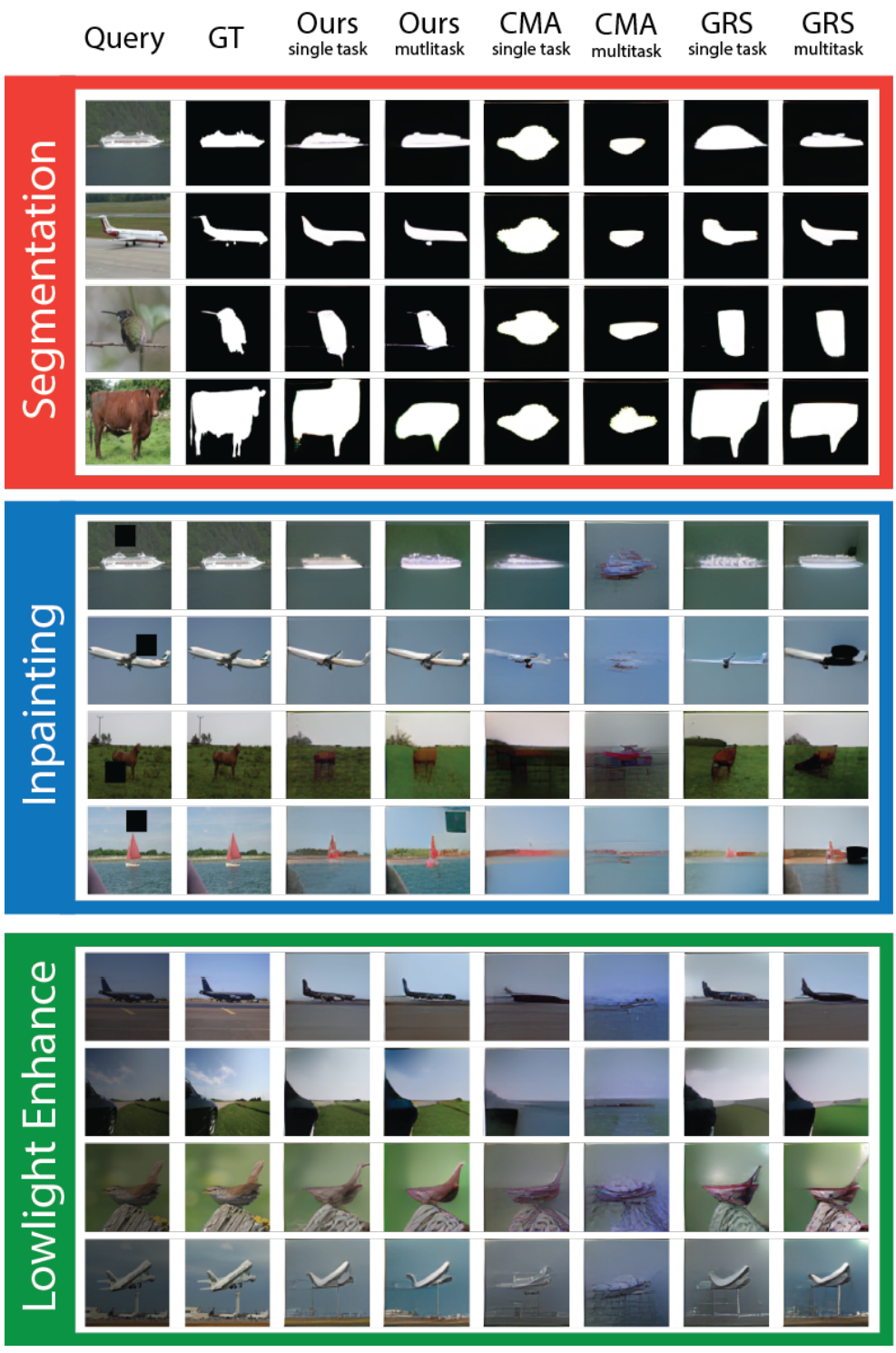

Figure 9. **Qualitative Examples.** We qualitatively compare the task-specific and multi-task variants of our methodology with the CMA and GRS baselines.

without restricting our ability to induce task implementation, and what is the best according p value? We ascertain this by searching for the optimal configuration to initialize the Greedy Random Search. We perform a grid search for k values from 14 to 20, and p values from 0.1 to 0.6 and report the evaluation metric for the Segmentation task on the batch of 10 images.

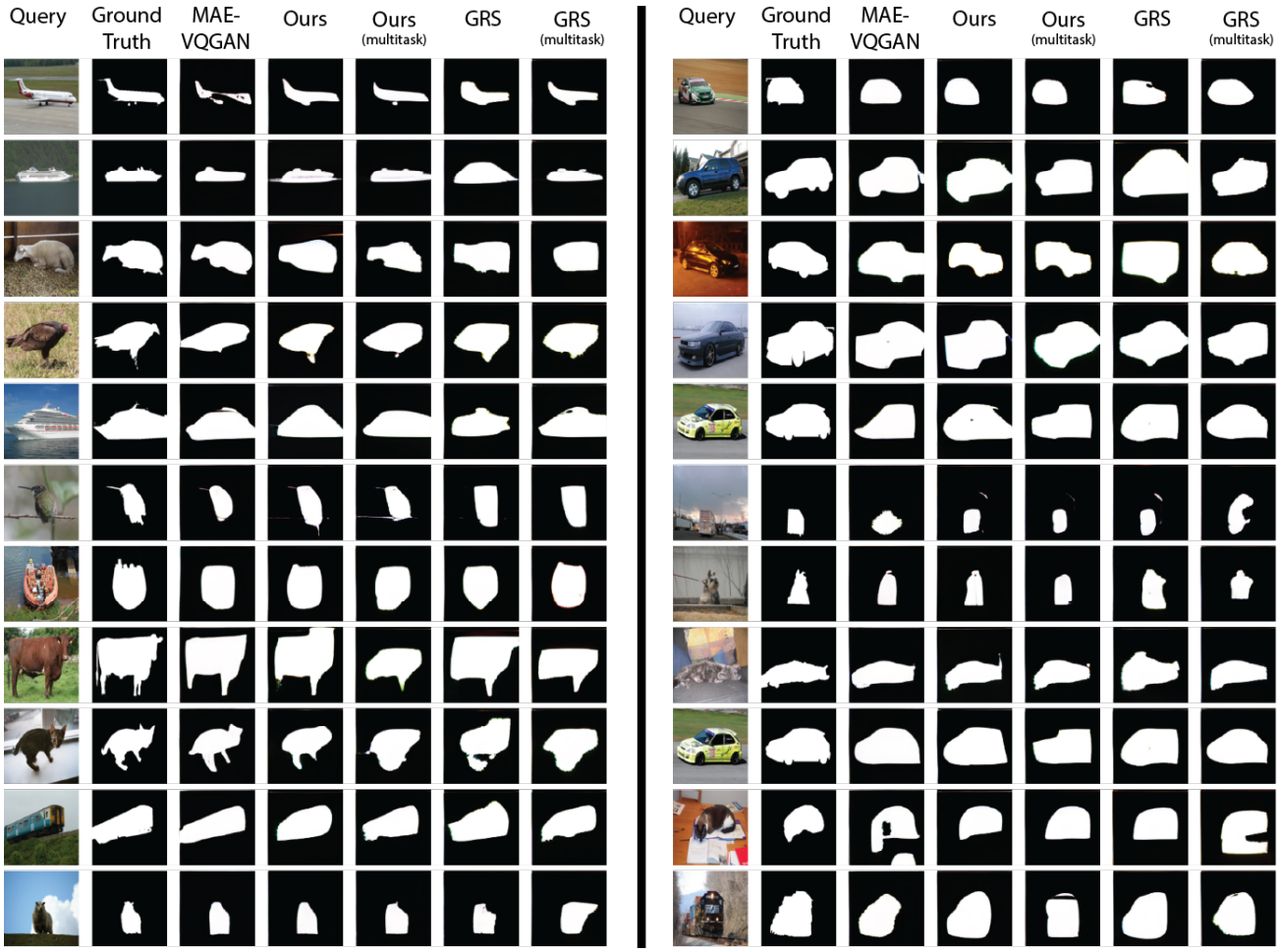

Figure 10. **Our Results on Segmentation Task**

Our goal is to find the (k, p) pair with highest performing random initialization.

Task Vectors Location in Encoder vs. Decoder. Similar to the ablation conducted for our main methodology (via REINFORCE (Williams, 1992)) implementation, we execute the Greedy Random Search for the Segmentation task by restricting interventions to the encoder only, the decoder only, and allowing for interventions throughout the whole network. It is key to note that in order to restrict interventions to the decoder only, which has 8 layers, the k value must be set to 8, whereas for isolating the encoder we can keep the original $k = 17$ value. We report the mIoU on the four splits for the Segmentation task seeking to find if interventions in both parts of the model are required for appropriate task implementation.

Patching Granularity. We execute our Greedy Random Search with three granularity levels, grouping by Quadrants, grouping by Heads, and grouping by Layers, and report the mIoU performance on the four splits for the Segmentation task.

13.3. GRS Implementation Experiments Results

Selecting K. We explore the optimal parameters for a random initialization. We find the best setup to be to constrain the search across the top $k = 17$ layers, sampling quadrants to patch with a probability of $p = 0.3$ (see Figure 14).

Task Vectors Location in Encoder vs. Decoder. We report the results on isolating the set of possible interventions to the encoder only, decoder only, in contrast to allowing interventions throughout the whole network. We can observe that in-context task learning builds upon both model components. The decoder, however, is more important. It is clear that intervening in both components is crucial for task implementation as we hypothesize that it is computed in a distributed

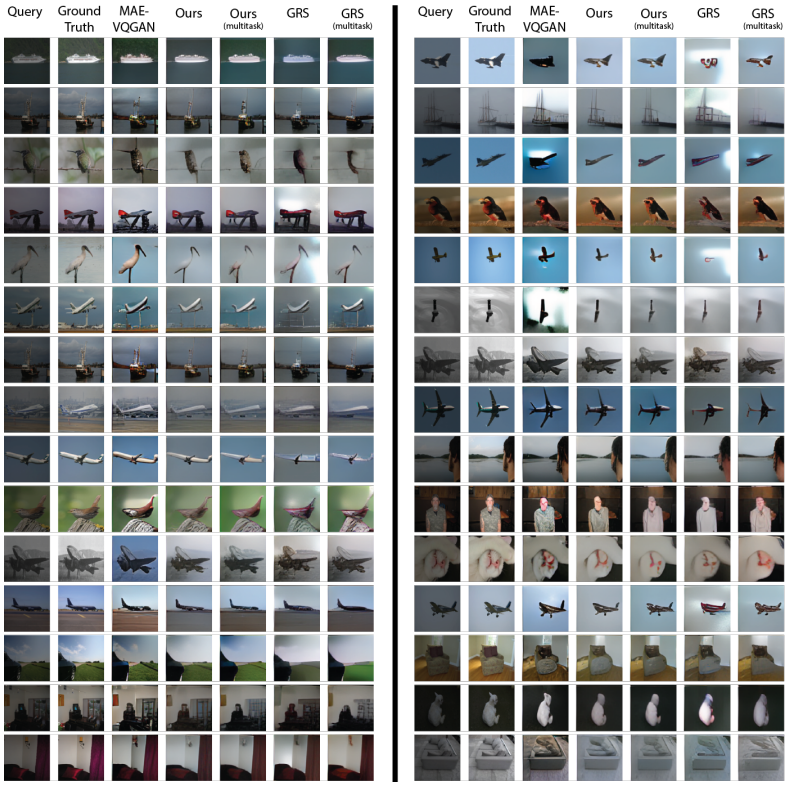

Figure 11. **Our Results on Lowlight Enhancement Task**

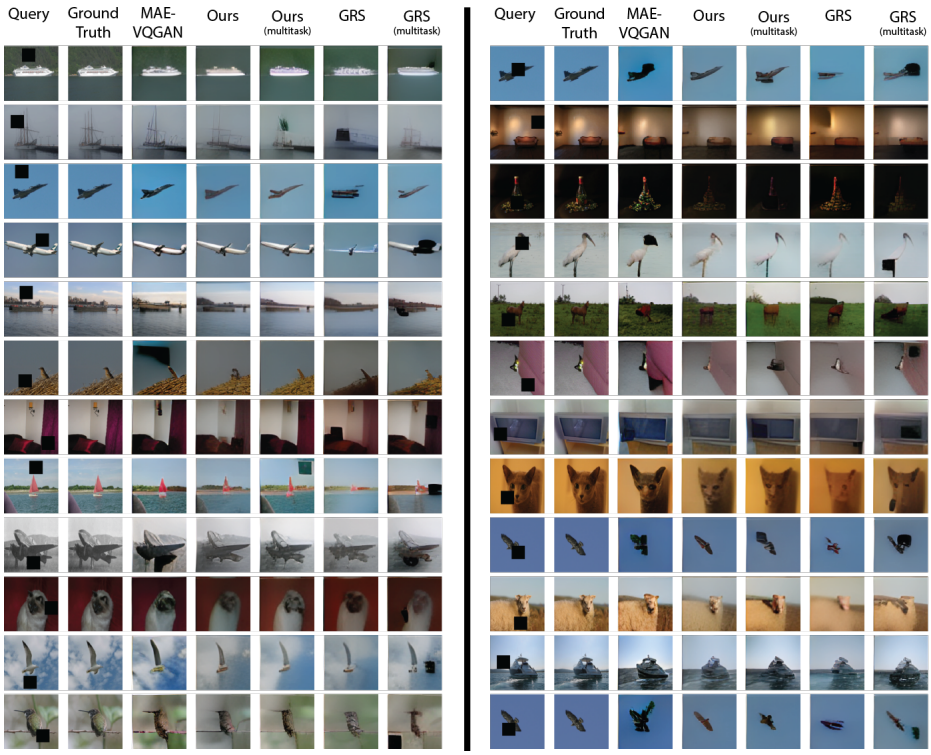

Figure 12. **Our Results on In-painting Task**

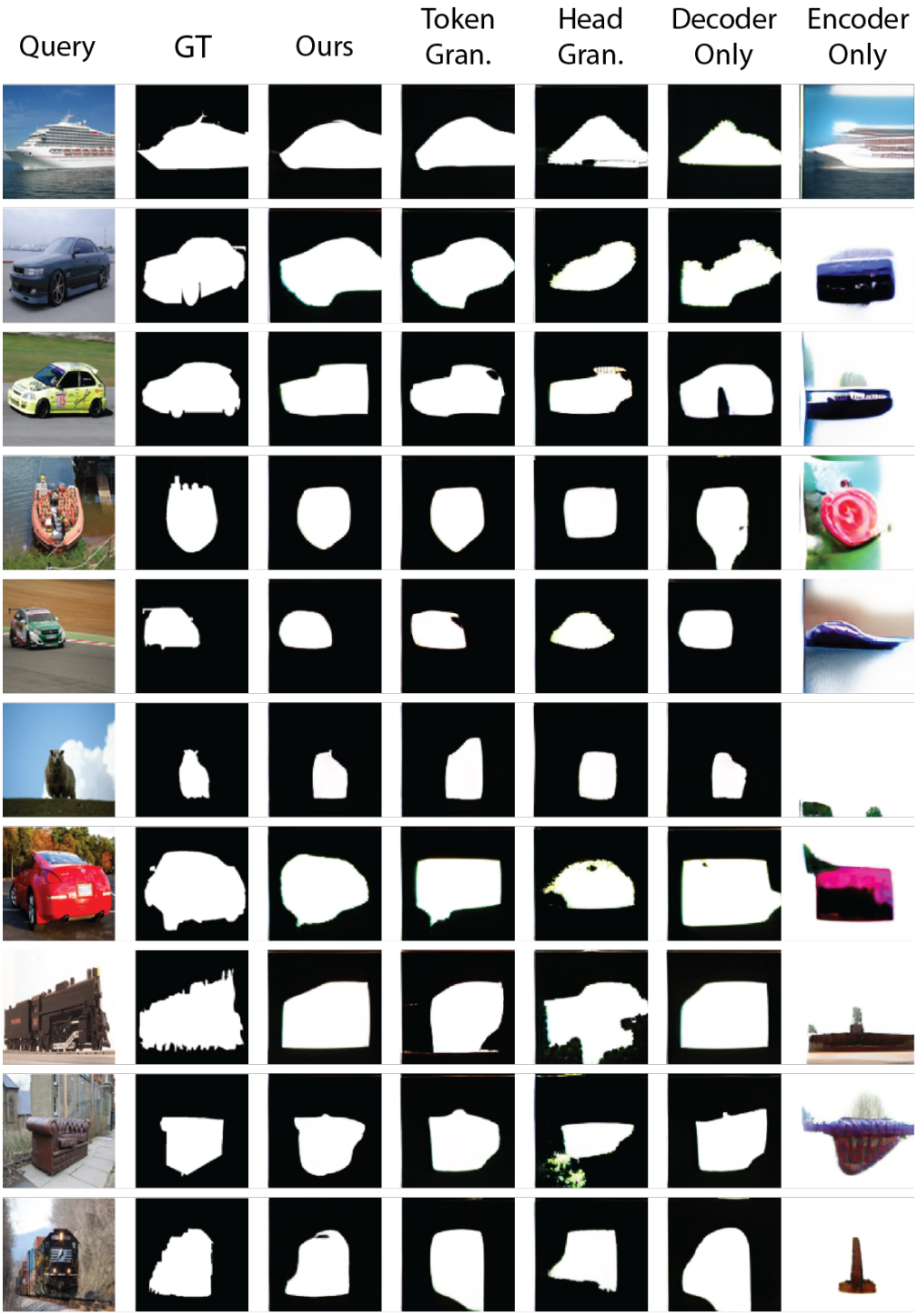

Figure 13. **REINFORCE Ablations for Segmentation Task**

fashion with cascading higher-order effects through the network where early interventions have strong downstream effects (see Table 10).

Patching Granularity. We explore the optimal granularity at which to group the tokens to reduce the dimensionality of the search space. Motivated by the emergence of quadrants in the per-token scoring visualization, and validated by

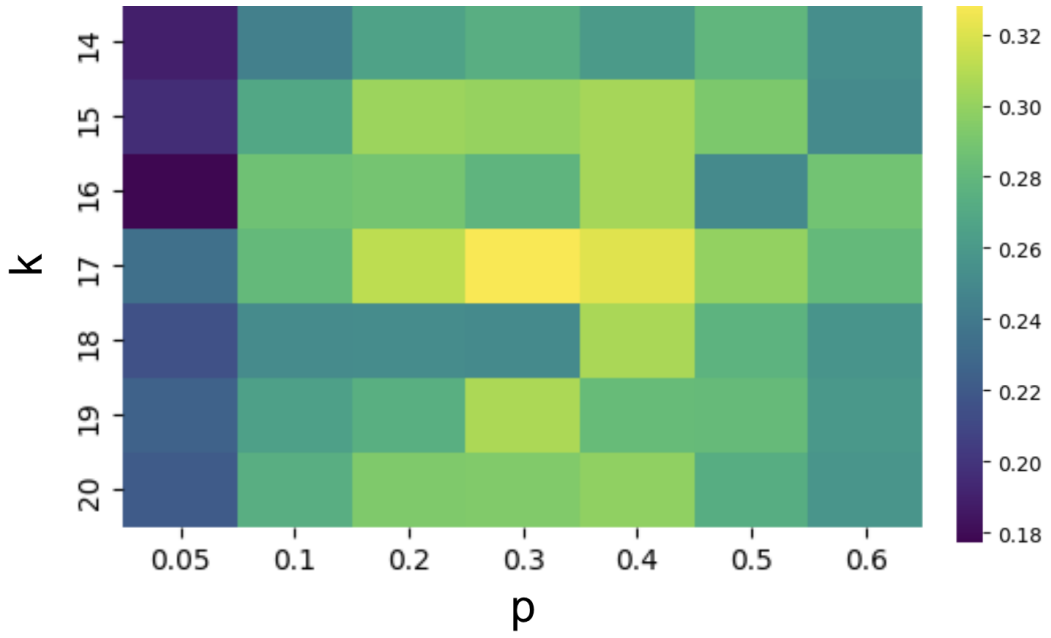

Figure 14. **Selecting Initialization Parameters.** We evaluate Foreground Segmentation mIoU on Pascal 5i using 10 images for different random initialization parameterized by K and p.

Table 10. **Isolating Task Locations.** Patching into Encoder only, Decoder only, and Both

	Segmentation ↑			
Model	Split 0	Split 1	Split 2	Split 3
GRS Both (Task-specific)	**0.33**	**0.35**	**0.30**	**0.30**
GRS Encoder (Task-specific)	0.13	0.22	0.20	0.20
GRS Decoder (Task-specific)	0.26	0.28	0.25	0.25

Table 11. **Optimal Patching Granularity.** Patching into Full Layers, Full Heads, or Quadrants only

	Segmentation ↑			
Model	Split 0	Split 1	Split 2	Split 3
GRS Quadrants (Task-specific)	**0.33**	**0.35**	**0.30**	**0.30**
GRS Heads (Task-specific)	0.15	0.15	0.14	0.13
GRS Layers (Task-specific)	0.28	0.31	0.26	0.27

attempting to group by whole attention heads (patching into all the tokens in the attention head) and group by whole layers (patching into all the attention heads of a layer), it is clear that quadrants provide the best trade-off between reducing the dimensionality of the search space and performance (see Table 11). It is interesting to note that patching into full layers reduces the search space to a size of 2^{32} whereas attention heads is 2^{512} and quadrants is 2^{768} for the encoder and 2^{384} in the decoder (disregarding top-k layer selection).

13.4. GRS Baseline Qualitative Comparisons

We provide a wider selection of examples comparing our GRS task vector patching methodology in comparison to **1) a selection of baselines, and 2) our ablations.** Alongside each figure, we accompany it with an according analysis.

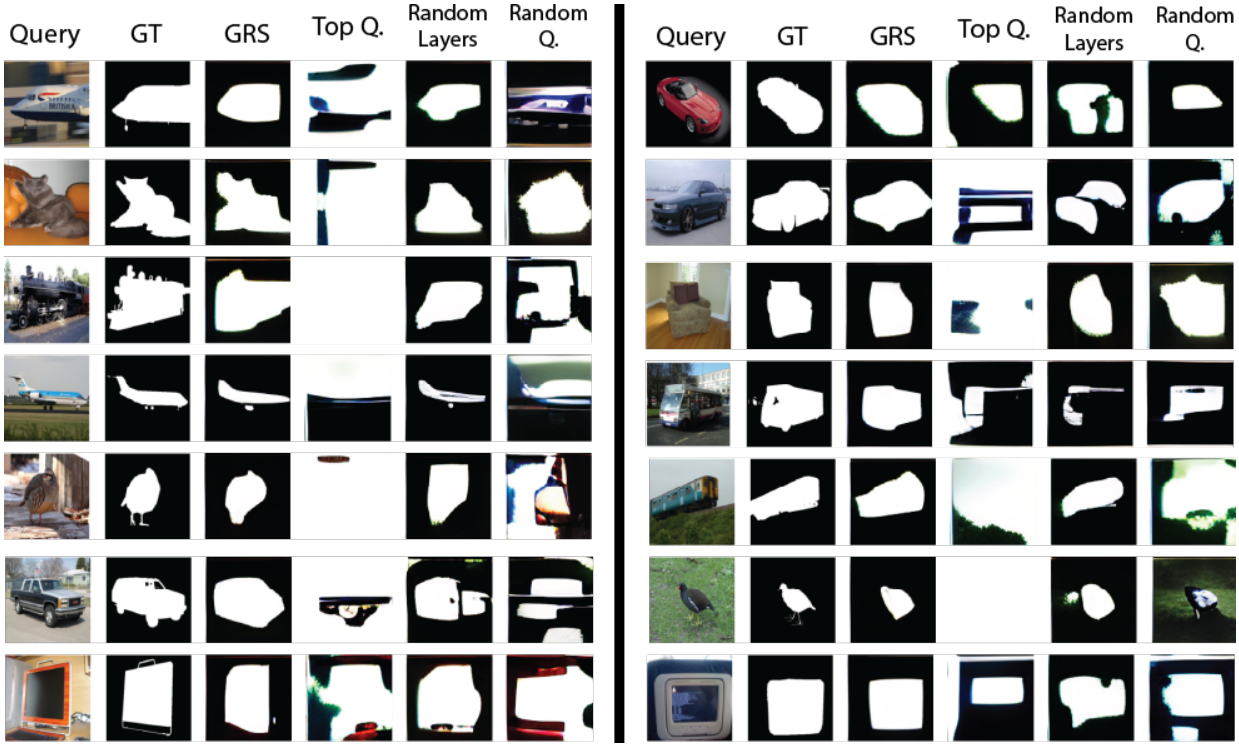

Figure 15. **GRS Baseline Comparison for Segmentation**

In the Figures 15, 16, and 17 we compare our GRS task-specific method with a handful of baselines defined in section 8.6. We have abbreviated Top Quadrants as Top Q, Random K Layers as Random Layers, and Random Quadrants as Random Q. It is clear that Top Quadrants struggles to output coherent completions. We believe this to be because of the need of patching into positions of different purposes other than task implementation such as positions that encode the input-output structure that a one-shot prompt provides. Further opportunities for exploration could include other scoring terms that take into account structural information provided by different prompt orientations. Random K Layers performs surprisingly well due to the efficiency of the Greedy Random Search but nonetheless does not reach the performance of using our scoring mechanism to select the top K layers. Finally Random Quadrants struggles to complete coherent results.

Finally, we present the qualitative analysis of the different ablations for the Segmentation task in Figure 18. Similarly to our main method (via REINFORCE (Williams, 1992)), it is clear that patching into decoder is more important than patching into the encoder but in the end, it is both parts in synchrony which report the best performance. Furthermore, it is clear that the optimal granularity for patching is at a quadrant level. We find it counterintuitive that layer-level patching performs better than head-level patching–as one would assume that a finer granularity provides better accuracies. However, we believe that by grouping per-layer we significantly reduce the search space (by a factor of 16) which reduces the probability of falling into a local optimum; whereas grouping by head, we suffer from a reduced precision but do not gain the benefits of a reduced search space magnitude (factor of 2 for encoder and factor of 3 for decoder when grouping by head instead of 16 when grouping by layer). Further exploration in this direction is of interest.

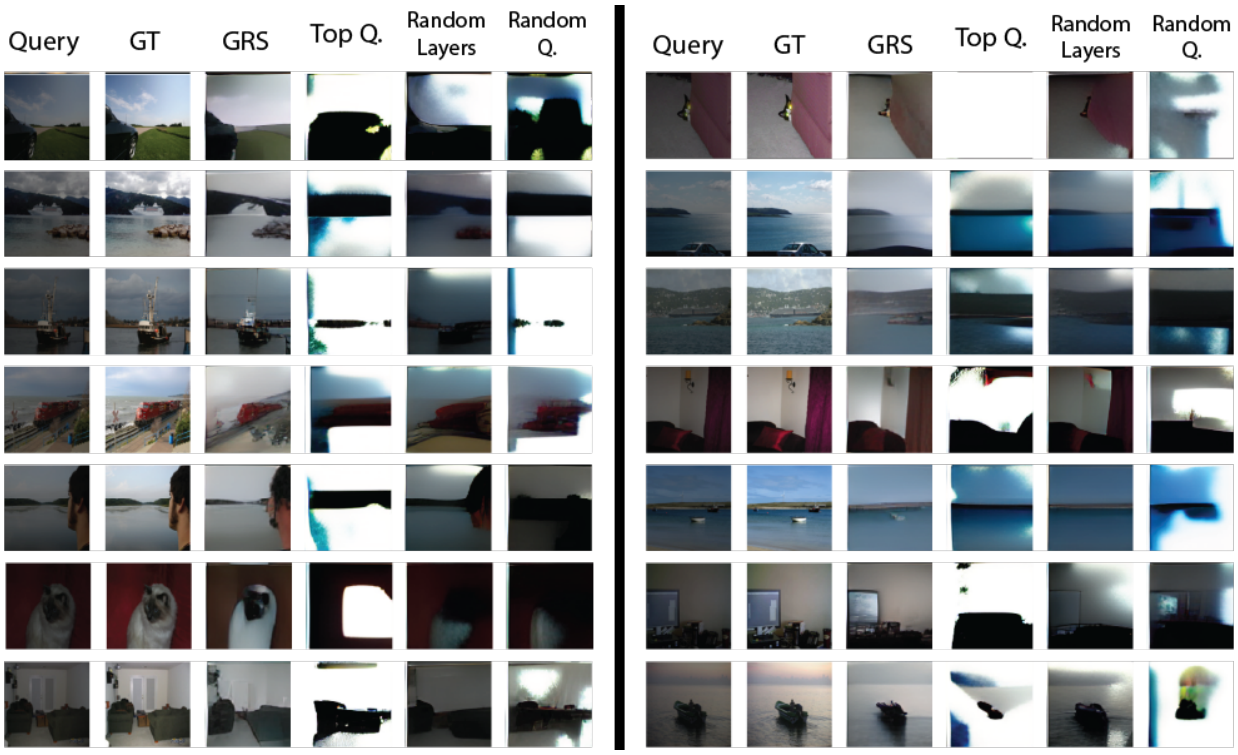

Figure 16. **GRS Baseline Comparison for Low light Enhancement**

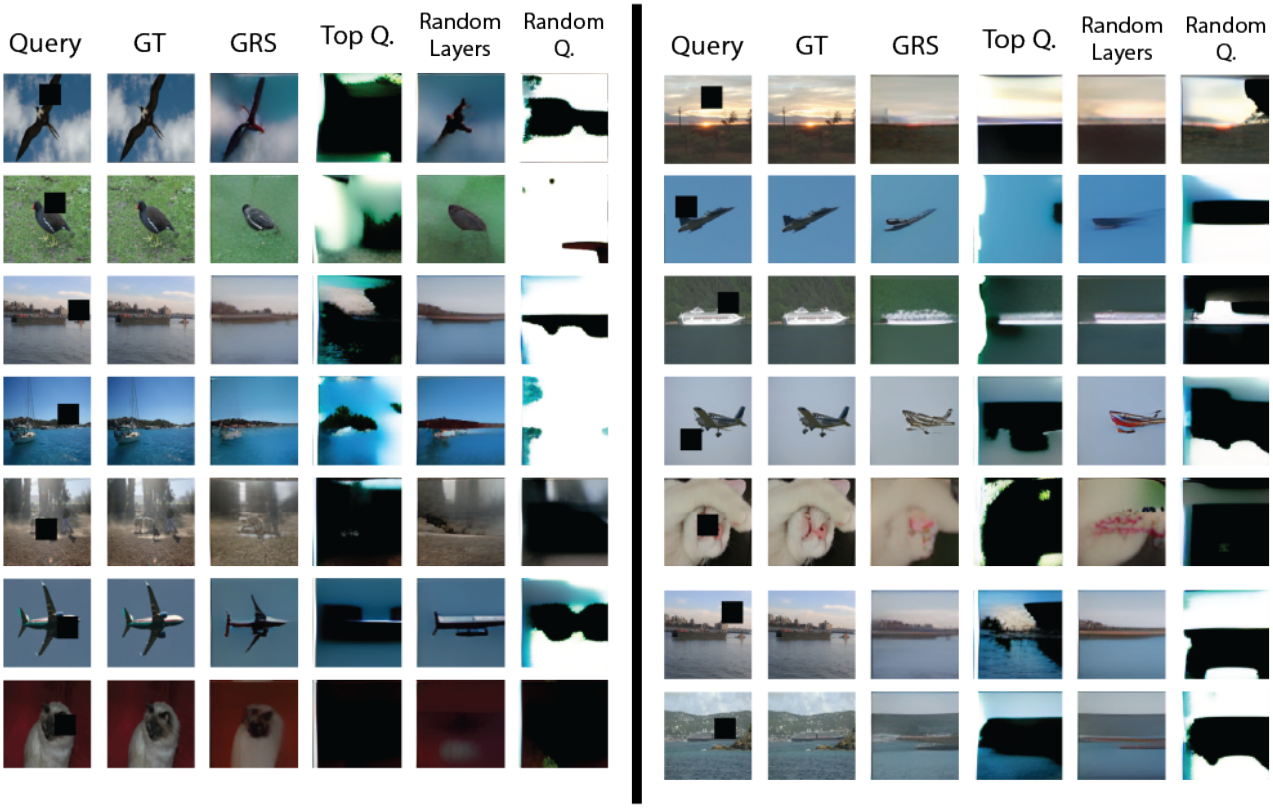

Figure 17. **GRS Baseline Comparison for In-painting**

Figure 18. **GRS Ablation for Segmentation Task**