RDAR: Reward-Driven Agent Relevance Estimation for Autonomous Driving

Carlo Bosio
UC Berkeley
c.bosio@berkeley.edu

Greg Woelki
Zoox Inc.
gwoelki@zoox.com

Noureldin Hendy
Zoox Inc.
nhendy@zoox.com

Nicholas Roy
Zoox Inc.
nroy@zoox.com

Byungsoo Kim
Zoox Inc.
bykim@zoox.com

Abstract: Human drivers focus only on a handful of agents at any one time. On the other hand, autonomous driving systems process complex scenes with numerous agents, regardless of whether they are pedestrians on a crosswalk or vehicles parked on the side of the road. While attention mechanisms offer an implicit way to reduce the input to the elements that affect decisions, existing attention mechanisms for capturing agent interactions are quadratic, and generally computationally expensive. We propose RDAR, a strategy to learn per-agent relevance — how much each agent influences the behavior of the controlled vehicle — by identifying which agents can be excluded from the input to a pre-trained behavior model. We formulate the masking procedure as a Markov Decision Process where the action consists of a binary mask indicating agent selection. We evaluate RDAR on a large-scale driving dataset, and demonstrate its ability to learn an accurate numerical measure of relevance by achieving comparable driving performance, in terms of overall progress, safety and performance, while processing significantly fewer agents compared to a state-of-the-art behavior model.

Keywords: Reinforcement Learning, Autonomous Driving, Agent Prioritization

1 Introduction

Humans, when driving, do not pay equal attention to all agents around them (e.g., other vehicles, pedestrians). Transfomer-based attention models offer the promise of attending only to relevant components of the input, but existing attention models are typically quadratic in the size of the input space. Driving models encounter hundreds of input tokens, leading to substantial computational complexity and latency [1, 2, 3].

In autonomous driving, there is a tension between the limited available compute resources and the desire to take advantage of scaling laws, large models, and test-time compute. Having access to numerical per-agent relevance scores would not only improve the interpretability of large driving models, but also allow compute resources to be prioritized for the features that are most important. In fact, when agents and other scene elements are represented explicitly as tokens, reasoning about interactions between these tokens (typically through self-attention or graph neural network operations) is quadratic and difficult to reduce using low-rank or other approximations that work well for long-sequence data. Reducing the number of tokens under consideration provides quadratic improvements in FLOPs used.

In this work, we introduce RDAR (Reward-Driven Agent Relevance), through which we quantify agent relevance through a learned approach. The basic intuition is that if an agent is not relevant towards the driving decisions of the controlled vehicle, then its absence would not change the controlled vehicle's driving behavior significantly. Thus, we quantify per-agent relevance by learning which agents can be masked out from the controlled vehicle's planner input while maintaining a good driving behavior. We formulate agent selection as a reinforcement learning (RL) problem

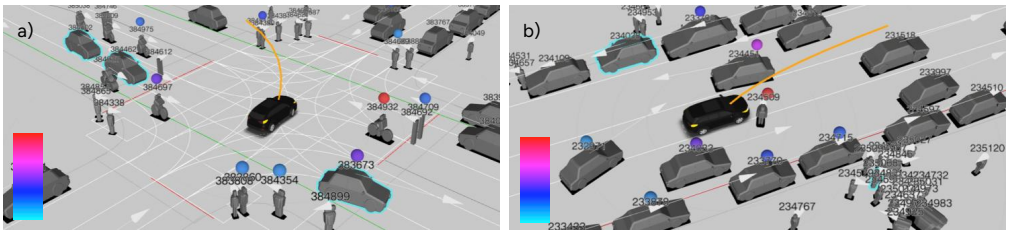

Figure 1: Example visualizations showing agent relevance assigned by our method. The top-k relevant agents are labeled with a colored dot hovering over them. As shown by the scale in the bottom left of each image, red corresponds to higher relevance and light blue corresponds to lower relevance. a) attending to cyclist while turning left, b) attending to pedestrian during stop. The controlled vehicle is in black.

where an action is a binary mask indicating which agents to include in the driving policy input, and which not to. The RDAR scoring policy is trained in the loop with a frozen, pre-trained driving policy and a simulator. At each time step, based on the relevance scores, an agent mask is fed to the driving policy, making the controlled vehicle blind to the lower score agents. As it will be clear from the following sections, this is not a binary classification problem over agents due to the underlying system dynamics (e.g., not observing an agent now could lead to a collision later) and to the unavailability of ground truth labels. Some examples of relevance (color-coded) computed by our method are shown in Fig. 1. Our main contributions are:

- A novel reinforcement learning formulation for agent relevance estimation;
- A sampling-based mechanism for agent selection that enables efficient training and inference;
- A comprehensive evaluation showing that we can maintain driving performance while processing only a handful of surrounding agents.

2 Related Work

Learning object ranking is a long-standing problem in deep learning and typically requires human-labeled data [4, 5, 6]. In fields such autonomous driving, a manual ranking process can be not straightforward, and require large amounts of labeled data. Input attribution [7] is a family of post-hoc analysis methods attempting to pinpoint which parts of the input are most responsible for a prediction. Attribution methods mostly focus on leave-one-out (LOO) approaches [8], where chunks of the input are individually removed, or masked, and are correlated with changes in model outputs.

Ranking agents in a driving scene based on their relevance is useful for both offline and online applications. Current autonomous driving systems quantify the relevance of surrounding agents either through fast, heuristic-based modules (e.g., based on euclidean distance), or learned models trained through supervised learning. Some approaches have been proposed for the supervision of these models, and they predominantly focus on LOO strategies coming from the attribution literature for agent prioritization [9], selective prediction [10], or offline introspection [11]. While these LOO approaches provide insights into individual agent contributions, they have some limitations. First of all, a change in driving behavior, captured by a shift in predicted action, represents a *different* driving behavior, but not necessarily a *worse* one. Second, these methods require multiple forward passes through a model (a planner in this case). Third, they do not capture the temporal dependencies caused by system dynamics. In this work, we propose a reward-driven method trained through reinforcement learning to estimate agent relevance.

RDAR computes per-agent relevance through just one forward pass, and is reward-driven instead of supervised through ground truth labels. Reinforcement learning (RL) approaches have shown promising results in autonomous driving applications [12], and are being integrated in production and deployed in real-world systems. Several works have shown large scale urban driving through the use of both RL and human-collected real world driving data [1, 13, 14]. In this work we build

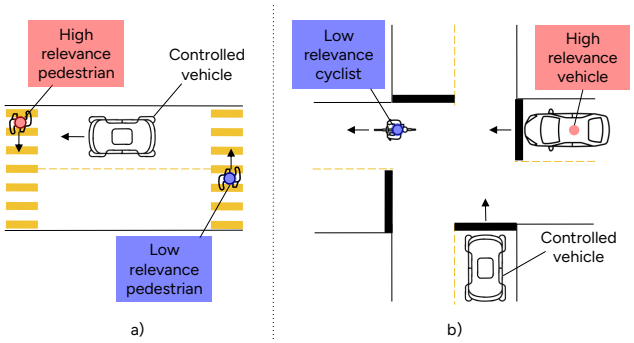

Figure 2: Example driving scenarios highlighting agent relevance. In a), the pedestrian crossing in front of the controlled vehicle is highly relevant, while the one behind is not. In b), the bike just went through the intersection and is not relevant anymore, while the car inching into the intersection is highly relevant.

upon an existing learned behavior model, and train a scoring policy with closed-loop RL through a novel formulation for agent selection.

3 Problem Setup

We wish to learn a policy π_θ^R assigning a *relevance score* to each agent in the driving scene based on its influence on the driving behavior of the controlled vehicle (θ denotes learnable parameters). We assume a pre-trained driving policy π^D mapping scene information to driving actions is available. We also assume that, associated with the policy π^D, there is a reward function r encoding some notion of good driving behavior. The RDAR scoring policy π_θ^R is trained in closed loop with the (frozen) driving policy π^D and a driving simulator.

Formally defining a notion of agent relevance is not straightforward. However, human drivers have a good intuitive concept of such notion, which allows them to pay selective attention to surrounding agents. With reference to Fig. 2a, a pedestrian crossing in front of the controlled vehicle is a highly relevant agent, because its presence means that the controlled vehicle must come to a stop and yield instead of driving through a crosswalk. At the same time, the pedestrian crossing behind the driver has low relevance. Similar considerations are true for the intersection scenario of Fig. 2b. The vehicle inching into the intersection has high relevance, because its presence means that the controlled vehicle must stop and yield. Instead, the cyclist who just passed through the intersection should not affect the controlled vehicle behavior. Therefore, we can say that an agent is relevant if hypothetically removing it from a driving scenario would cause the controlled vehicle to have a different behavior.

Markov Decision Process formulation Following the above intuition, we formulate the agent relevance estimation problem as a Markov Decision Process (MDP). The policy π_θ^R outputs per-agent relevance scores, which can be interpreted as logits of a categorical distribution. If agent i is sampled from this *relevance distribution*, it gets processed by the driving policy π^D, otherwise it is masked out and ignored by π^D. Given a hyperparameter $k \in \mathbb{N}$, an action is then a subset of k surrounding agents, or a k-sample, to be processed by π^D. Our goal is thus to learn π_θ^R such that the return is maximized in expectation. Inaccurate relevance scores would make the driving policy blind to important agents in the scene, leading to low reward behaviors (e.g., collisions). The MDP setup for this process is defined by a standard tuple (S, A, r, P, μ_0), where:

- S is the state space, including the controlled vehicle state, surrounding agent states (expressed in the controlled vehicle reference frame), road network and route information (see Fig. 3c);
- $A = \{0, 1\}^N$ is the action space, consisting of binary masks of size N (number of agents) indicating which agents to include in the planning input. The logits of the action distribution are the relevance scores (details in the following sections);

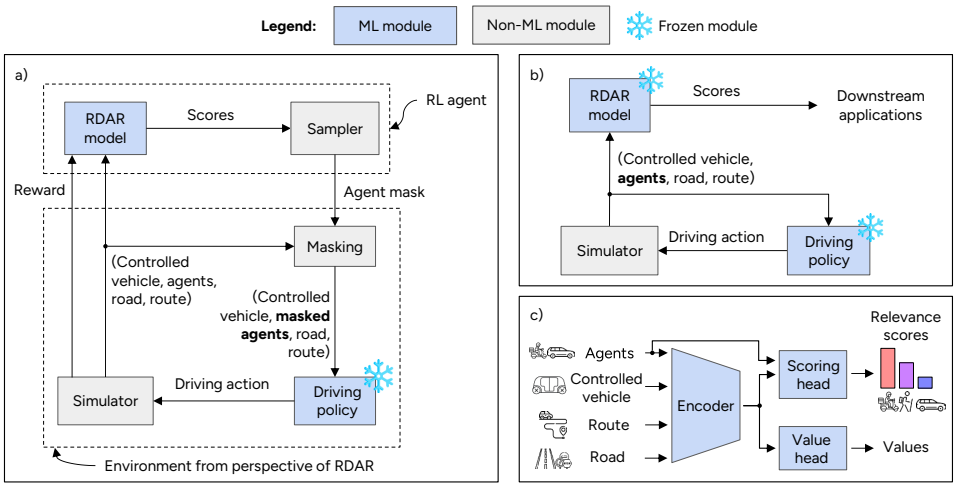

Figure 3: a) Block diagram of the system at training time. b) Block diagram of the system at deployment time. c) RDAR model structure.

- r is the reward function encoding good driving behavior. This is ideally the same reward or scoring function accompanying π^D;
- P is the transition probability function associated to the environment. Note that the environment, from the perspective of the reinforcement learning agent π^R_θ, consists of π^D and the actual driving environment altogether (see Fig. 3a);
- μ_0 is the initial state distribution.

The nature of the actions space makes this problem similar to a contextual multi-armed bandit (CMAB) [15], with the subtle difference that in this case the action changes the context, which in CMABs is assumed to be independent of the action.

Agent Selection Mechanism At each time step t, the RDAR policy π^R_θ outputs a vector of per-agent relevance scores $\phi_t = [\phi_1, \phi_2, \ldots, \phi_N]$. At deployment time, the top-k scoring agents are selected, while at training time, agents are randomly sampled to encourage exploration. For sampling, the scores are converted to a categorical distribution $p_t = [p_1, p_2, ..., p_N]$ over the binary agent selection action space through a softmax, where

$$p_i = \frac{\exp(\phi_i)}{\sum_{j=1}^{N} \exp(\phi_j)}. \tag{1}$$

Drawing one sample from this distribution corresponds to selecting one agent. If we draw exactly one sample, the probability of agent i being selected is p_i. We can thus get a k-sample by drawing k samples sequentially, without replacement, by renormalizing the probabilities at each step. We denote an agent k-sample as

$$a = (a_1, a_2, \ldots, a_k), \quad a_i \in \{1, ..., N\}, \tag{2}$$

where each component a_i is the integer index corresponding to the selected agent. Note that this notation and an N-dimensional binary vector are equivalent. Then, the probability of selecting an ordered sample of agents without replacement is

$$P(a_1, \ldots, a_k) = \prod_{i=1}^{k} \frac{p_{a_i}}{1 - \sum_{j=1}^{i-1} p_{a_j}}, \tag{3}$$

where the denominators are the renormalization terms. Note that although we describe the sampling process as sequential, the Gumbel top-k trick enables efficient, single-step sampling without replacement [16, 17]. The trick consists of perturbing the distribution p_t with a Gumbel distribution, and greedily selecting the top-k elements:

$$(a_1, ..., a_k) = \arg\text{top-}k \ \{ p_t - \log(-\log U) \}, \quad U \sim \text{Uniform}(0, 1) \tag{4}$$

By applying the logarithm to (3), we can compute the log-likelihood of the k-sample:

$$\log P(a_1, \ldots, a_k) = \sum_{i=1}^{k} \log p_{a_i} - \sum_{i=1}^{k} \log\Big(1 - \sum_{j=1}^{i-1} p_{a_j}\Big). \tag{5}$$

Since the scores are the output of our model π_θ^R, (5) is exactly what we need for policy gradient updates in an RL framework. Once a k-sample is picked, only those k agents are processed by the driving policy π^D. The action output by the driving policy is then applied to the simulator, and the overall state is updated. The simulator also produces the reward signal r_t for RDAR. The process is then repeated.

Reinforcement learning framework We train the policy using an off-policy actor–critic framework with V-trace corrections [18]. The loss function combines four components:

$$\mathcal{L} = \mathcal{L}_{\text{policy}} + \lambda_c \mathcal{L}_{\text{critic}} + \lambda_e \mathcal{L}_{\text{entropy}} + \lambda_s \mathcal{L}_{\text{smoothing}}. \tag{6}$$

The four components are respectively policy gradient loss, critic loss, entropy regularization loss, and action smoothing loss. Their exact expressions are:

$$\mathcal{L}_{\text{policy}} = -\mathbb{E}_t \left[\rho_t \, \log \pi_\theta^R(a_t \mid s_t) \, \hat{A}_t^{\text{v-trace}} \right], \tag{7}$$

$$\mathcal{L}_{\text{critic}} = \mathbb{E}_t \left[\left(V_\theta(s_t) - V_t^{\text{target}} \right)^2 \right], \tag{8}$$

$$\mathcal{L}_{\text{entropy}} = \mathbb{E}_t \left[-\sum_{i=1}^{N} \pi_\theta^R(i \mid s_t) \log \pi_\theta^R(i \mid s_t) \right], \tag{9}$$

$$\mathcal{L}_{\text{smooth}} = \mathbb{E}_t \left[\sum_{i=1}^{N} \| \pi_\theta^R(i \mid s_t) - \pi_\theta^R(i \mid s_{t-1}) \|^2 \right], \tag{10}$$

where $\rho_t = \frac{\pi_\theta^R(a_t|s_t)}{\mu(a_t|s_t)}$ are clipped importance weights, a_t is the agent k-sample at time step t as in (2). The $\hat{A}_t^{\text{v-trace}}$ and V_t^{target} terms are computed following [18]. The log-likelihood term in (7) is computed as in (5). The entropy and smoothing loss components, are calculated on the logits directly and do not depend on the k-sample a_t. The entropy component favors uniformity in the relevance scores, and therefore encourages exploration. The action smoothing component encourages the scores corresponding to the same agent to be consistent across time. It is possible that fewer than N agents are physically present in a scene at a given time, in which case the loss terms corresponding to non-existing agents are masked. Finally, λ_c, λ_e, λ_s are hyperparameters weighing the loss terms, selected empirically.

4 Implementation

Architecture The RDAR model architecture consists of three main components. An encoder module processes the full scene context (controlled vehicle state, surrounding agent states, road and route information) and computes embeddings. There are two heads: a scoring head, mapping embeddings to agent relevance scores, and a value head, approximating the value function. Note that the value function for the relevance scoring policy has a different meaning than the value function for the driving policy, since the expectation is over all possible k-samples rather than all possible driving actions. The three options for the scoring head make use of embedding of varying depth from the encoder. The first option (Fig. 4a) just uses the features from the agent projection layer. In this case, only agent state information is fed to the scoring head. The second option (Fig. 4b) uses the embeddings output by the agent encoder module. In this case, the embeddings also encode agent interactions. The third option (Fig. 4c) uses the output from the scene encoder, and reprojects it back to the agent level through a transformer block. In this case, all information from the driving scene is used to compute agent relevance. The value head is kept the same for the three architectures, and uses all available information.

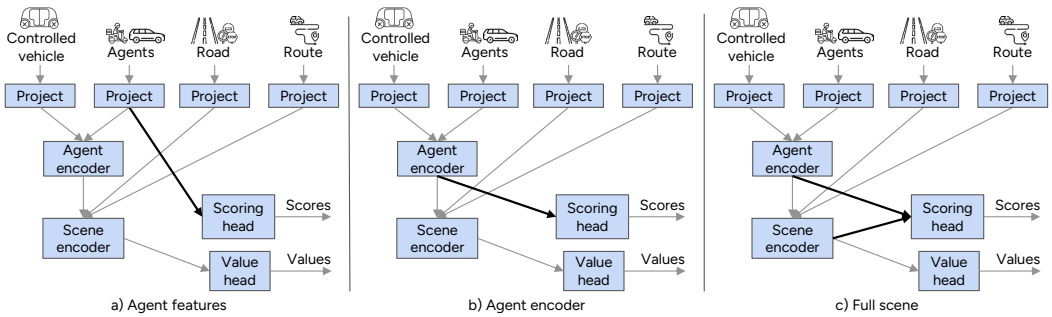

Figure 4: Different model architectures tested. Left: the scoring head consumes directly agent features projected through a linear layer. Middle: the scoring head consumes the output of the agent encoder, a transformer block cross attending among agents. Right: the scoring head consumes full scene information encoded into a latent embedding from the scene encoder, and attending back to agent tokens.

Training details The training-time agent sampling procedure described in Section 3 is done through the Gumbel top-k trick [17], natively used, for instance, in the JAX's implementation of the categorical distribution [19]. Also at training time, the number k of agents sampled is randomized to make sure the model learns actual relevance scores and not only to differentiate between top-k and non top-k agents. We uniformly sample a different value k per driving scenario. To achieve scale, we use a distributed, asynchronous reinforcement learning infrastructure similar to IMPALA [18]. We found these hyperparameter values to give the best performances: learning rate $2 \cdot 10^{-5}$, $\lambda_c = 0.1$, $\lambda_e = 0.2$, $\lambda_s = 0.05$. No sampling happens at deployment time, and the top-k agents are selected greedily (analogous to selecting the argmax action in standard RL).

5 Experimental Setup

Datasets We train and evaluate our approach on large-scale proprietary datasets consisting of ten-second long scenarios of real world, diverse urban driving. The training dataset contains around two million scenarios, while the evaluation dataset contains twelve thousand.

Metrics To quantitatively evaluate our method, we use standard driving metrics which we compute on the 12k scenario evaluation set. In these evaluations, the candidate relevance scoring policies and baselines are used in closed-loop as filters, with the driving policy π^D processing only a subset k of all the agents present at any one time. At evaluation, we select the top-k scoring agents greedily (analogously to selecting an action through argmax in standard RL). We use the following metrics:

1. *Collisions [%]*: percentage of scenarios in which a collision occurs (lower is better);
2. *Traffic light [%]*: percentage of scenarios in which a traffic light is violated (lower is better);
3. *Stop line [%]*: percentage of scenarios in which a stop line is skipped (lower is better);
4. *Off-road [%]*: percentage of scenarios in which the vehicle drives off-road (lower is better);
5. *Comfort*: metric combining four motion aspects – forward/backward acceleration, turning acceleration, and how suddenly or abruptly these accelerations change. These are weighted, averaged, then converted to a 0-1 score where 1 means smooth driving and 0 means jerky, uncomfortable motion (higher is better).
6. *Progress ratio*: relative progress along the route with respect to the ground truth human log;
7. *Complexity*: computation required by the scoring method as a function of the number N of agents in a scene.

Baselines We compare RDAR to other scoring strategies. The evaluation is done in closed-loop, using these strategies to pick the top-k agents to be processed by π^D:

- Closest-k selection: Select the k closest agents to the controlled vehicle – equivalent to agents' relevance scores being inversely proportional to their distance to the controlled vehicle;

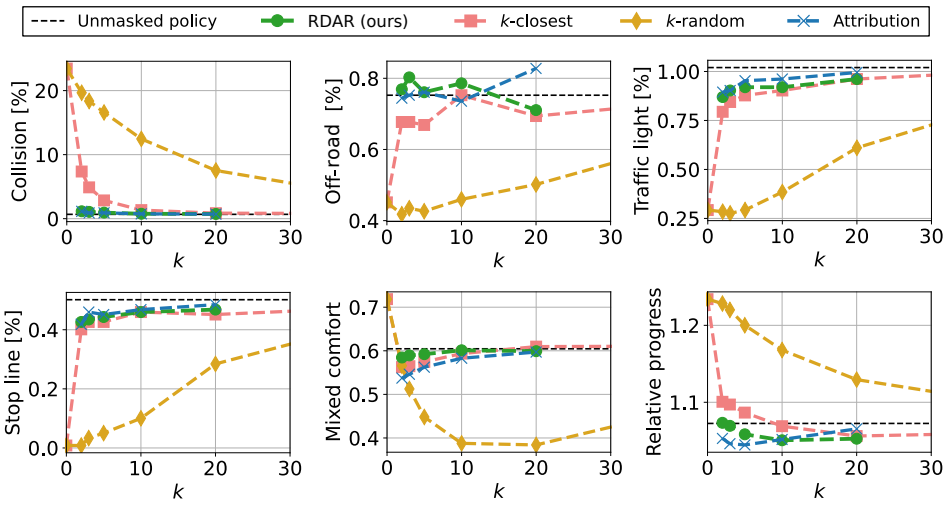

Figure 5: Metrics of best RDAR model and baselines against number k of selected agents in closed-loop. All the closed-loop metrics for RDAR are close to the full policy baseline, supporting the claim that we are able to learn true agent relevance. In this case, the RDAR model is the version processing full scene information (see Fig. 4c).

- Random-k selection: Randomly select k agents from the scene – equivalent to agents' relevance scores being drawn uniformly at random;
- Attribution-based scoring: scores obtained via input attribution [11]. At each time step, the procedure is as follows: the pre-trained driving policy π^D is evaluated $N + 1$ times (one with full scene information and one with each individual agent omitted in turn). For each agent, the Jensen–Shannon divergence between the action from its masked-out pass and the nominal full-scene pass is computed.

The overall performance of these baselines when varying k is shown in Fig. 5. When $k = N$ no agents are masked out from the driving policy.

6 Results

Quantitative results The three different architectures proposed in Fig. 4 show comparable performances, and the model using full scene information leads to the least collisions. We report plots showing the trends of the reference metrics when varying the value k in Fig. 5. We can see that RDAR has comparable performances to the attribution method, and requires only a fraction of the computation time. RDAR is also able to drive with a fraction of percent performance regressions compared to the nominal, full policy, while processing an order of magnitude fewer agents. Table 1 shows the actual numbers relative to the $k = 10$ case. It is interesting to see that the random scoring policy outperforms all the others in when it comes to rules of traffic (off-road, traffic lights, stop lines). This comes, as expected, at the cost of much higher collision rates. Scores computed using only agent features (Fig. 4a) or attending to the control vehicle state (Fig. 4b), which achieves good closed-loop driving, requires fewer FLOPs compared to using full scene information (Fig. 4c). On the other hand, full scene information enables enhanced awareness and lower collisions.

Qualitative results We also report some visualizations from the same closed-loop evaluation rollouts (Fig. 6). The scenes represented are challenging, cluttered driving scenes in which incorrect relevance quantification would lead to wrong masking and bad behaviors. The top-k agents have a colored dot hovering over them, which is color-coded based on the actual score. These examples are from our best full-scene scoring policy. We can see that the agent importance assigned by our model is aligned with human intuition.

7 Discussion and Conclusions

Our model provides insights into which agents influence driving decisions, enabling better understanding of planner behaviors. It can also inform how to allocate costly computation in a principled

	Collisions [%]	Off-road [%]	Traffic light [%]	Stop line [%]	Comfort	Rel. progress	Complexity
RDAR, Agent features (Fig. 4a)	0.94	0.71	0.97	0.44	0.57	1.06	$O(1)$
RDAR, Agent encoder (Fig. 4b)	0.89	0.70	0.89	0.43	0.57	1.06	$O(1)$
RDAR, Full scene (Fig. 4c)	0.77	0.79	0.92	0.46	0.60	1.05	$O(1)$
Closest-k	1.34	0.75	0.90	0.46	0.59	1.07	$O(1)$
Random-k	12.5	0.46	0.38	0.10	0.39	1.17	$O(1)$
Attribution	0.75	0.74	0.96	0.47	0.58	1.05	$O(N)$
Baseline (no filter)	0.68	0.75	1.02	0.50	0.61	1.07	–

Table 1: Closed-loop metrics of different methods with $k = 10$ agents. Using only agent features or reasoning about agent interactions leads to good closed-loop driving performance, and requires fewer FLOPs compared to full scene, which on the other hand achieves lower collision rate.

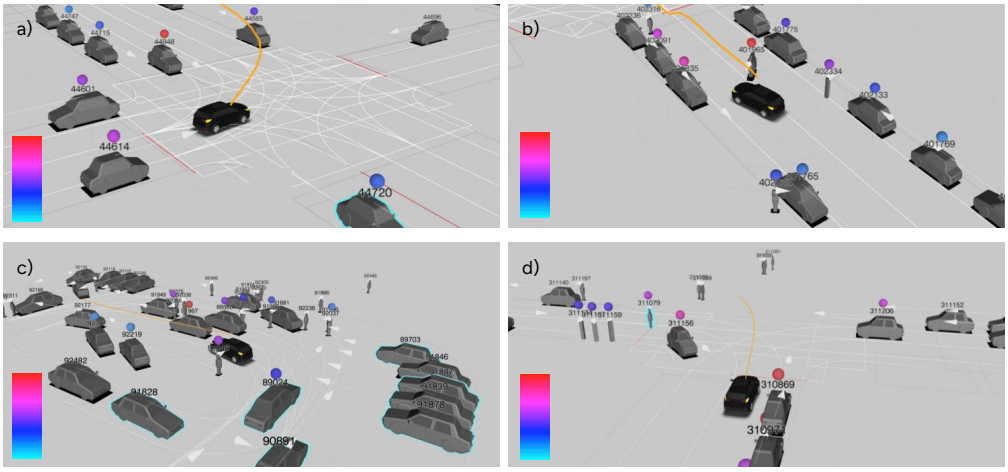

Figure 6: Example visualizations showing agent relevance assigned by our method. The top-k relevant agents are labeled with a colored dot hovering over them. As shown by the scale in the bottom left of each image, red corresponds to higher relevance and light blue corresponds to lower relevance. The controlled vehicle is in black. Agent importance seems to be aligned with human intuition.

way–for example, running joint trajectory prediction or computing vision embeddings only for the most relevant agents.

This work opens several interesting directions. First, similar relevance-scoring methods could be applied to other components of the driving scene, such as road information. A challenge here is the potential for distribution shifts when masking inputs; in our case such effects are mild, since driving scenarios remain in-distribution regardless of the number or position of agents, but investigating mitigation strategies is important. Second, the scoring policy's action space could be expanded beyond masking. Instead of excluding agents, the prioritization module could be trained to trigger targeted computation on selected agents, such as expensive vision embeddings, so that enhanced representations directly translate into downstream gains like improved driving rewards.

We introduced a reinforcement learning approach to estimate agent relevance in driving scenarios. By formulating relevance scoring as an agent-masking MDP, we enable end-to-end training of a scoring policy with a driving policy in the loop. Our method avoids costly post-hoc attribution and repeated forward passes, making it well suited for real-time autonomy stacks. In closed-loop evaluation, we show that comparable driving performance can be achieved while processing an order of magnitude fewer agents, highlighting the benefits of our approach in terms of behavior model introspection and dynamic compute allocation.

Acknowledgments

The authors would like to thank Anubhav Paras and Ethan Pronovost, Paul Viola for their insightful comments and advice.

References

[1] M. Harmel, A. Paras, A. Pasternak, N. Roy, and G. Linscott. Scaling is all you need: Autonomous driving with jax-accelerated reinforcement learning. *arXiv preprint arXiv:2312.15122*, 2023.

[2] X. Huang, E. M. Wolff, P. Vernaza, T. Phan-Minh, H. Chen, D. S. Hayden, M. Edmonds, B. Pierce, X. Chen, P. E. Jacob, et al. Drivegpt: Scaling autoregressive behavior models for driving. *arXiv preprint arXiv:2412.14415*, 2024.

[3] M. Baniodeh, K. Goel, S. Ettinger, C. Fuertes, A. Seff, T. Shen, C. Gulino, C. Yang, G. Jerfel, D. Choe, et al. Scaling laws of motion forecasting and planning–a technical report. *arXiv preprint arXiv:2506.08228*, 2025.

[4] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *Advances in neural information processing systems*, 10, 1997.

[5] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

[6] K. G. Jamieson and R. Nowak. Active ranking using pairwise comparisons. *Advances in neural information processing systems*, 24, 2011.

[7] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[8] F. Liu, N. Kandpal, and C. Raffel. Attribot: A bag of tricks for efficiently approximating leave-one-out context attribution. *arXiv preprint arXiv:2411.15102*, 2024.

[9] K. S. Refaat, K. Ding, N. Ponomareva, and S. Ross. Agent prioritization for autonomous navigation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2060–2067. IEEE, 2019.

[10] E. Tolstaya, R. Mahjourian, C. Downey, B. Vadarajan, B. Sapp, and D. Anguelov. Identifying driver interactions via conditional behavior prediction. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3473–3479. IEEE, 2021.

[11] M. Cusumano-Towner, D. Hafner, A. Hertzberg, B. Huval, A. Petrenko, E. Vinitsky, E. Wijmans, T. Killian, S. Bowers, O. Sener, et al. Robust autonomy emerges from self-play. *arXiv preprint arXiv:2502.03349*, 2025.

[12] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE transactions on intelligent transportation systems*, 23(6):4909–4926, 2021.

[13] C. Gulino, J. Fu, W. Luo, G. Tucker, E. Bronstein, Y. Lu, J. Harb, X. Pan, Y. Wang, X. Chen, et al. Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research. *Advances in Neural Information Processing Systems*, 36:7730–7742, 2023.

[14] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, R. Roelofs, B. Sapp, B. White, A. Faust, S. Whiteson, et al. Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7553–7560. IEEE, 2023.

[15] T. Lu, D. Pál, and M. Pál. Contextual multi-armed bandits. In *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*, pages 485–492. JMLR Workshop and Conference Proceedings, 2010.

[16] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[17] W. Kool, H. Van Hoof, and M. Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International conference on machine learning*, pages 3499–3508. PMLR, 2019.

[18] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

[19] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.