IEEE Robotics and Automation Letters (RAL) paper presented at
2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

MotionPerceiver: Real-Time Occupancy Forecasting for Embedded Systems

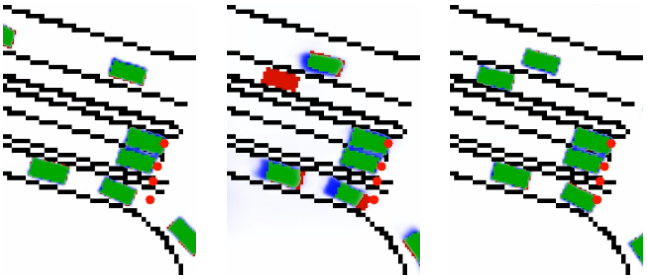
Bryce Ferenczi[1†], Michael Burke[1], Tom Drummond[1,2]

***Abstract*— This work introduces a novel and adaptable architecture designed for real-time occupancy forecasting that outperforms existing state-of-the-art models on the Waymo Open Motion Dataset in Soft IOU. The proposed model uses recursive latent state estimation with learned transformer-based functions to effectively update and evolve the state. This enables highly efficient real-time inference on embedded systems, as profiled on an Nvidia Xavier AGX. Our model, MotionPerceiver, achieves this by encoding a scene into a latent state that evolves in time through self-attention mechanisms. Additionally, it incorporates relevant scene observations, such as traffic signals, road topology and agent detections, through cross-attention mechanisms. This forms an efficient data-streaming architecture, that contrasts with the expensive, fixed-sequence input common in existing models. The architecture also offers the distinct advantage of generating occupancy predictions through localized querying based on a point-of-interest, as opposed to generating fixed-size occupancy images that render potentially irrelevant regions.**

I. Introduction

Motion forecasting is a crucial step in trajectory planning for autonomous vehicles, integral to optimal decision making and preventing collisions with other agents in a dynamic environment. This task, however, is fraught with complexities, owing to the multifaceted factors that influence the trajectories of other agents. These factors encompass the environmental context, per-agent goals, navigable area and social interactions between agents. Compounding this challenge is the heterogeneity of the data structures derived from the sensor suite of autonomous vehicles. This data spans a spectrum of temporal attributes, ranging from time-sensitive (e.g. the position of other agents or traffic signals) to time-invariant (e.g. lane markings or pedestrian crossings). This data diversity creates challenges around representation, as it can manifest in sparse and concise forms, resembling a singular point in the environment, or more intricate forms, exemplified by lane markings encoded as directional splines. Furthermore, the volume of data within a scene varies significantly, influenced by instance-based features, such as agents and lane markings. Practical use within autonomous systems introduces a number of additional requirements: real-time capability, efficient query processes for downstream algorithms, and uncertainty estimation to enable risk-based motion planning.

[1]Department of Electrical and Computer Systems Engineering, Monash University, Australia
[2]Department of Computing and Information Systems, Melbourne University, Australia
[†] Corresponding Author: bryce.ferenczi@monash.edu



(a) Last Observation ($t = 0s$) (b) Before Next Observation ($t = 0.4s$) (c) Next Observation ($t = 0.5s$)

Fig. 1. MotionPerceiver implements recursive state estimation for motion forecasting. Here, an initial scene observation (a) at $t = 0s$ is forecast (b) $0.4s$ into the future. This learned prediction begins to accumulate error and uncertainty for targets in motion, and shows no occupancy for an agent that was not initially detected. Frame (c) shows the predicted occupancy after a scene observation has applied an update to the latent state at $t = 0.5s$. Agent occupancy is refined, without explicit data association, and a new agent added to the state. Images are color coded green $\rightarrow$ true positive (occupancy prediction > 0.5), blue $\rightarrow$ false positive, red $\rightarrow$ false negative, black $\rightarrow$ rasterized road graph, red dots $\rightarrow$ traffic signals.

This paper introduces MotionPerceiver, a real-time motion forecasting model explicitly designed to handle these representation challenges, seamlessly integrating information from a broad range of sensors. At its core, MotionPerceiver models a scene as a latent state that continually evolves in time with updates derived from latent representations of scene observations as they become available. Importantly, this architecture uses a design cue from Perceiver-IO [1], using fixed latent state dimensions to facilitate deterministic memory and computational cost. MotionPerceiver leverages a learned time evolution function to predict the future latent state, rooted in multi-head self-attention mechanisms to capture dynamic interactions among features embedded in the latent state, such as vehicles and pedestrians. In order to assimilate incoming data, such as agent positions and traffic signals, we employ cross-attention to facilitate information transfer to the projected latent state. This scheme can be considered analogous to a recursive state estimation process, improving the predicted latent scene representation as novel information emerges (Fig. 1). Predicting occupancy is performed by querying the latent state at the desired time, using an encoding of the desired query position. For a comprehensive occupancy prediction of the scene, a grid of positions can be used to generate a rasterized birds-eye-view image.

Motion forecasting necessitates operating at a rate surpassing real-time to incorporate new observations of the scene. To efficiently process a causal signal, it is prefer-

able to only process new incoming information, rather than redundantly reprocessing old data. Algorithms that use a fixed-sequence for input are inefficient since they begin inference from scratch and reprocess historical information. Unfortunately, this is the dominant paradigm used by the majority of motion forecasting architectures ([2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15]). A more streamlined approach would be to augment the prior motion forecast with new information. Furthermore, scalability is also paramount, to cope with a large numbers of agents. Solutions that scale super-linearly or are not parallelisable are impractical for real-world deployment. MotionPerceiver's latent state prediction-update scheme is designed with a computationally-deterministic data streaming paradigm in mind. This allows the previous computational investment in building the latent state representation to be retained, while seamlessly incorporating new measurements.

Finally, robustness to noisy or incomplete data is essential for real-world deployment where occlusions and sensor blind spots are common. Sequence-based models commonly use agent tracklets as input, which can be corrupted during occlusions or a faulty tracking algorithm that outputs switched ids. MotionPerceiver's latent state update does not use temporal data association at the input, and is robust to this problem. In summary, the core contributions of this work are:

- A novel transformer-based architecture that uses recursive state estimation to forecast occupancy using agent detections and seamlessly integrates contextual information.
- The first data-streaming based motion forecasting model that reports real-time inference on an edge device.

This model outperforms existing models in predicted occupancy (Soft IoU) on the Waymo Open Motion dataset.

II. Related Work

Motion forecasting is the task of predicting the future position of agents in a scene. There are two reference frames which are commonly used, agent-centric and scene-centric. Agent-centric methods [7] transform the scene into each agent's point-of-view for prediction. This paradigm can become prohibitively expensive due to inference cost scaling per agent in the scene. Scene-centric methods holistically encode the entire scene and jointly predict motion of all agents. This is the dominant architecture in the literature, and the paradigm followed by MotionPerceiver. A variety of scene-encoding architectures have been explored in this category including: point-pillars [6], [16], [9], fully-convolutional [7], [11], transformer [8], graph [5] and hybrid methods [4].

Trajectory forecasting involves predicting continuous splines that represent the future trajectory for each agent. This can take the form of a probabilistic set, representing a multi-modal decision distribution of the target [3], [7], [13], [14]. A variety of higher-level architectural designs have been explored to accomplish this task. State-of-the-art trajectory planners commonly follow a paradigm of finding a set of likely goals, and generate feasible trajectories towards these [3], [17]. Anchors are also used as a foundation for multi-modal prediction forecasting. Anchors can be derived statistically from the dataset [18] or learned during training [14]. Other methods forecast future trajectories without using anchors or goals as grounding [7], [10], [13], [15]. An important formalisation is to predict control inputs for the motion model of the agent to prevent physically infeasible predictions [13], [14]. The weakness of per-agent trajectory forecasting is the high complexity this formulation can elicit in larger, crowded scenes. A path-planning algorithm has to parse, optimize and validate a multi-modal trajectory distribution for each agent, adding a large number of constraints.

Agent-based Occupancy forecasting on the other-hand, is the goal of predicting if a point in space will be occupied by an agent in the future. This paradigm is more amenable to path planning algorithms as they can check whether a position in the navigable area is occupied with a single query. This formulation, in isolation, erases the identities of agents. Methods which also predict the flow of occupancy over time [4], [5], [6], [8], [9], [11] enable retracing from where occupancy has originated, and the agent responsible. Occupancy forecasting implicitly permits multi-modal predictions as the occupancy attributed to a particular agent is able to spread beyond the real size of the agent as its position becomes more uncertain further in the future. Occupancy prediction is performed by MotionPerceiver due to its ease-of-use in downstream planning or trajectory optimisation algorithms.

Sensor-based Occupancy Forecasting estimates birds-eye-view (BEV) or volumetric occupancy of a scene using time-of-flight sensors such as Lidar and/or Radar. This contrasts with the object conditioned occupancy forecasting considered by MotionPerceiver. Intrinsic challenges to this task are sparse-to-dense reconstruction, and reasoning around sensor-occluded areas of the scene. A common approach is to predict occupancy from a Lidar point-cloud observation, forecast to the next observation's space-time location, then enforce consistency with the next point-cloud [19], [20], [21]. Radar introduces additional challenges due to various noise modalities such as reflection and receiver saturation. Despite these challenges, its longer range, penetration and lower cost make it an ideal candidate for use in autonomous vehicles. Hence, [22] develop a model to predict occupancy from Radar, using accompanying Lidar supervision for training.

Including scene context is an important capability for forecasting models, especially for predicting over longer time horizons. Context can be invariant over time – this typically includes topographical information such as lane markings, potholes or signage. Time-sensitive context includes dynamic entities such as traffic signalling equipment and other agents in the environment. Motion forecasting models within autonomous driving settings predominantly use lane markings as context [3], [5], [23], [8], [17]. Lane marking information, represented as directed splines, are rarely used as raw input to a model. Preprocessing into a more amenable representation is needed. Commonly, lane markings are rasterized into an image that can be processed by a CNN for feature extraction [8], [7]. Alternatively, VectorNet [24] uses a hierarchical graph neural network for encoding. Individual splines are ini-

tially processed into features of fixed dimension, then global interactions are modeled between these features, resulting in a concise representation of each spline as a feature of fixed dimension. A smaller subset of models include traffic signals [4], [6], [9], [10], but this is likely attributed to some popular datasets [25], [26] not containing high quality traffic signal information and its limited quantitative effect on performance. MotionPerceiver is explicitly designed to be extendable and integrate a variety of features that may influence agent dynamics, which is important as new contextual features become available in driving datasets.

The vast majority of forecasting models consume a **fixed sequence** of observations for inference due to the simplicity of capturing spatio-temporal features as a single input. The most common method is concatenating the history of an agent into a feature-vector [24], [5], [15]. A small number of methods accumulate timestamped positions of agents in a point-cloud [6], [9]. In general, these sequence based models have the disadvantage that inference is started anew for each forecast when a new scene observation is captured. Furthermore, accurate tracking and data association methods are required to build the history for each agent (a tracklet). Motion forecasting performance could be potentially degraded if malformed tracklets are introduced by tracking algorithm errors. MotionPerceiver uses a streaming based approach that avoids repetition of inference and use of tracklets. Here, we update our forecast as new observations come in, based on our previous forecast.

III. Preliminaries

A. Transformer attention

The multi-head attention function of the **transformer** [27], denoted as MHA, has become pervasive in deep learning. The output of each attention head, $\mathbf{H}_i, i \in \{1, \ldots, h\}$, is a weighted sum of $\mathbf{V}_i$, using learned correlations between queries $\mathbf{Q}_i$ and keys $\mathbf{K}_i$,

$$\mathbf{H}_i = \mathrm{Attn}(\mathbf{Q}W_i^q, \mathbf{K}W_i^k, \mathbf{V}W_i^v) = \mathrm{softmax}(\frac{\mathbf{Q}_i\mathbf{K}_i^T}{\sqrt{d_k}})\mathbf{V}_i, \tag{1}$$

where each attention head uses a different projection of inputs $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$, using learned parameters $W_i^q \in \mathbb{R}^{d\times d_q}$,$W_i^k \in \mathbb{R}^{d\times d_q}$,$W_i^v \in \mathbb{R}^{d\times d_v}$. For clarity, d is the dimension of the model output, d_q is the dimension of $\mathbf{K}$ and d_v is the dimension of $\mathbf{V}$. The result from each head is linearly combined using learned parameter $W^o \in \mathbb{R}^{hd_v\times d}$

$$\hat{\mathbf{H}} = \mathrm{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{concat}(\mathbf{H}_1, \ldots, \mathbf{H}_h)W^o. \tag{2}$$

A typical transformer block includes a residual pass through a multi-layer perceptron (mlp), to produce final output $\mathbf{H}'$,

$$\mathbf{H}' = \mathrm{mlp}(\hat{\mathbf{H}}) + \hat{\mathbf{H}}. \tag{3}$$

We use transformer blocks as the foundation of each MotionPerceiver function to model the interactions between features in the latent state and observations from the scene.

B. Sinusoidal Position Encoding

Transformer models are often paired with a positional encoding scheme, enabling better identification of distance-based correlations between tokens. Common positional features in motion forecasting include coordinates $(x, y) \rightarrow \mathbf{p} \in \mathbb{R}^2$ and pose $(x, y, \theta) \rightarrow \hat{\mathbf{p}} \in \mathbb{R}^3$, where x and y denote position, and θ represents heading in a reference frame. The sinusoidal position encoding scheme uses a series of f_n evenly spaced frequency components f_i, from f_{min} to f_{max}. To encode a scalar position p, each element is calculated

$$\mathcal{P}_i(p) = \begin{cases} \sin(f_i\pi p) & i < f_n \\ \cos(f_i\pi p) & else, \end{cases} \tag{4}$$

and concatenated to form the sinusoidal encoding vector,

$$\mathcal{P}(p) = \mathrm{concat}(\mathcal{P}_1(p), \ldots, \mathcal{P}_{2\cdot f_n}(p)) \tag{5}$$

For a vector, $\hat{\mathbf{p}}$, this procedure is repeated for each element,

$$\mathcal{P}(\hat{\mathbf{p}}) = \mathrm{concat}(\mathcal{P}(\hat{\mathbf{p}}_x), \mathcal{P}(\hat{\mathbf{p}}_y), \mathcal{P}(\hat{\mathbf{p}}_\theta)). \tag{6}$$

IV. MotionPerceiver

A. Problem formulation

Our goal is to make predictions of the future occupancy within a region of interest, conditioned on observations that include the positions, orientations and velocities of other agents in a scene, traffic signalling information, and other contextual information such as road lane markings. We assume that agent id is not available, and the number of agents observed can vary over time, due to occlusions, or agents entering and leaving the scene. We consider a streaming data paradigm, where sensors are providing these observations periodically, in real time.

B. Tokenizing Observations

A key advantage of the MotionPerceiver architecture is adaptability to ingest diverse data sources, needing only a reasonable strategy to encode information into a set of N_o tokens with dimension C_o, $\mathbf{I_t} = \{I_t^0, \ldots, I_t^{N_o}\}, I_t^n \in \mathbb{R}^{C_o}$. In this section, we outline the method used for transforming raw agent, traffic signal and road-graph observations into a form MotionPerceiver can consume. Using a scene region-of-interest of $160 \times 160m$, position is normalized to $[-1, 1]$ and sinusiodally encoded with (6). We use $f_{min} = 1Hz$, $f_{max} = 320Hz$ and f_n is specified per-feature, dependent on desired fidelity.

A number of properties from the agent observation are used to construct the tokenized representation used by MotionPerceiver. Observed pose $\hat{\mathbf{p}}$, is sinusoidally encoded with $f_n = 64$. Vehicle dimensions, (h, w), are incorporated to rasterize a correctly sized occupancy mask. Rate-of-change $\dot{\hat{\mathbf{p}}}$, enables identification of vehicles in motion with one observation. These features are concatenated to produce a vector representation of each visible agent $A = (\mathcal{P}(\hat{\mathbf{p}}), \dot{\hat{\mathbf{p}}}, h, w)$ with dimension C_a. Hence, for inference, a variable number of agents N_a, observed at time t, are transformed into a set of tokens $\mathbf{A}_t = \{A^1, \ldots, A^{N_a}\}_t$.

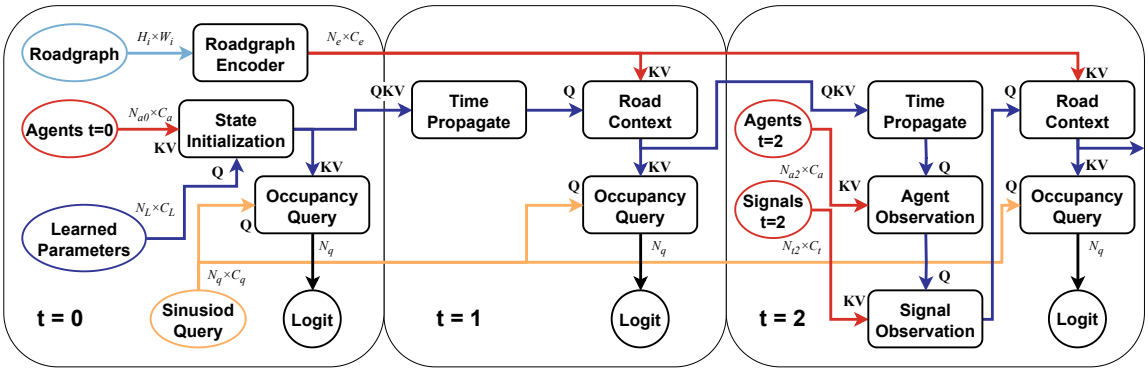


Fig. 2. An illustration of the use of MotionPerceiver's architecture for real-time occupancy prediction. At $t = 0$, the latent state is initialised with the first observation of agents in the scene. The evolution of this latent state is shown using dark blue arrows. Tokenized observations from the scene (light red) are queried by the latent state for information. Rasterized road-graph context (light blue) can be encoded once and provide contextual information at each time-step. The latent state can be queried with a position (light orange) to receive an estimate of occupancy probability at each time-step. When there is no observation information ($t = 1$), the latent state is simply propagated forward in time and updated with road-graph context. This is the operation used for forecasting future occupancy or interpolation between observations. At time-steps when scene observations are available ($t = 2$), the latent state queries the observation for information, reducing accumulated errors and adding newly observed agents. Dimensions $N_x \times C_x$ describes the number of tokens N_x and channels per token C_x. Additional inference diagrams can be found at https://sites.google.com/monash.edu/motionperceiver.

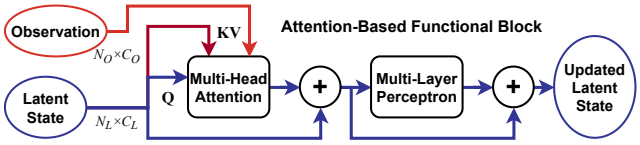


Fig. 3. Self- and cross-attention is used to apply changes to the latent state. The latent state (dark blue), is always used as the query. Self-attention sources the key-value from the latent state (dark red), communicating information between the N_L variables in the latent state. Cross-attention uses the observation as the key-value (light red) for the latent state to query and transfer information from the observation to the latent state.

To construct a traffic signal token, observed position $\mathbf{p}$ is sinusoidally encoded with $f_n = 32$ and concatenated with a one-hot encoding of the signal type $\mathcal{T}$ [1], producing feature vector $T = (\mathcal{P}(\mathbf{p}), \mathcal{T})$ of dimension C_t. For N_t traffic signals observed at time t, we create a set $\mathbf{T}_t = \{T^1, \ldots, T^{N_t}\}_t$.

A rasterization strategy is used to tokenize road-graph information for MotionPerceiver. Lane markings are rasterized as a $H_i \times W_i$ image and passed through a convolutional neural network (CNN), creating a $H_p \times W_p$ feature image. Each pixel value $\hat{R}$, is a high-dimensional representation of a section of the map. This is concatenated with a sinusoidal encoding ($f_n = 16$) of its location $\mathbf{p}$, creating a feature vector $R = (\hat{R}, \mathcal{P}(\mathbf{p}))$ of dimension C_e. Hence, from a variable number the lane markings in a scene, we create a fixed set of tokens $\mathbf{R} = \{R^1, \ldots, R^{N_e}\}$ where N_e is the number of pixels $H_p \times W_p$. This decouples run-time cost from the density of road-graph content in the scene[2].

C. Recursive State Estimation Using Transformers

MotionPerceiver, shown in Fig. 2, models agents navigating a dynamic social environment. We learn how a representation of the scene, described as a set of latent variables $\mathbf{S} = \{S_1, \ldots, S_{N_L}\}, S_n \in \mathbb{R}^{C_L}$, evolves in time (shown in the dark blue path of Fig. 2). Self- and cross-attention based functions allow the model to capture and express social dynamics between the agents in the latent state and context when predicting the future.

To initialize the latent state, $\mathbf{S}$, a set of learned parameters, query the first observation of agents, $\mathbf{A}_0$, with cross-attention. This is followed by 6 self-attention applications. This forms our *StateInitialization* module, depicted in Fig. 2. From here, we begin performing recursive state estimation with MotionPerceiver.

The time evolution process modeled as learned function $\mathcal{F}$, only depends on the previous state $\mathbf{S_{t-1|\cdot}}$, to predict the next state $\mathbf{S_{t|t-1}}$, akin to a Markovian dynamics model,

$$\mathbf{S_{t|t-1}} = \mathcal{F}(\mathbf{S_{t-1|\cdot}}). \tag{7}$$

Here, we adopt Kalman filter notation, $\mathbf{S_{t|t-1}}$, to represent a state at time t, given observations at time $t-1$. The latent state $\mathbf{S_{t-1|\cdot}}$ is used for the arguments in (2),

$$\mathbf{Q} = \mathbf{S_{t-1|\cdot}},\ \mathbf{K} = \mathbf{S_{t-1|\cdot}},\ \mathbf{V} = \mathbf{S_{t-1|\cdot}}, \tag{8}$$

creating a self-attention between the N_L variables in the latent state. This process is visualized in the dark blue and dark red path(s) of Fig. 3 where $\mathbf{S_{t-1|\cdot}}$ is the "Latent State" input. This self-attention function is repeated 6 times in a block to create the *TimePropagate* module in Fig. 2.

To transfer information from tokenized observation $\mathbf{I_t}$ to the latent state, a learned update function $\mathcal{U}$ is used **after** the state has been propagated to the time frame when the information was captured,

$$\mathbf{S_{t|t}} = \mathcal{U}(\mathcal{F}(\mathbf{S_{t-1|\cdot}}), \mathbf{I_t}) = \mathcal{U}(\mathbf{S_{t|t-1}}, \mathbf{I_t}). \tag{9}$$

Here, the latent variables of $\mathbf{S_{t|t-1}}$ query the key-value pairs derived from $\mathbf{I_t}$ using (2),

$$\mathbf{Q} = \mathbf{S_{t|t-1}}, \quad \mathbf{K} = \mathbf{I_t}, \quad \mathbf{V} = \mathbf{I_t}. \tag{10}$$

[1]Examples of signal types include green straight and red turn.

[2]Creating a rasterized image of splines is relatively inexpensive.

This cross-attention process is depicted in Fig. 3 and follows the light red path for "Observation" $\mathbf{I_t}$, and the dark blue path for "Latent State" $\mathbf{S_{t|t-1}}$. This is used in the $RoadContext$, $AgentObservation$ and $SignalObservation$ modules. An additional self-attention is applied after cross-attention in the $RoadContext$ and $SignalObservation$ modules.

The latent state can be queried with a set of positions $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{N_q}\}$ to yield predicted occupancy logits, $\hat{o_t}$, given the current state of the scene $\mathbf{S_{t|\cdot}}$ with learned emission function $\mathcal{O}$,

$$\hat{o_t} = \mathcal{O}(\mathbf{P}, \mathbf{S}_{t|\cdot}) = Pr(\mathbf{P}|\mathbf{S_{t|\cdot}}). \tag{11}$$

Once again, this is performed using cross-attention (2),

$$\mathbf{Q} = \mathcal{P}(\mathbf{P}), \quad \mathbf{K} = \mathbf{S_{t|\cdot}}, \quad \mathbf{V} = \mathbf{S_{t|\cdot}}. \tag{12}$$

A potential weakness of point-wise methods is missing near-field context. If greater accuracy is desired, a region of pixels can be queried and decoded with a small CNN (Conv Decode). Importantly, this does not nullify our "local query" capability, we still query a patch rather than the full scene.

Intuitively, these operations can be considered analogous to a recursive Bayesian estimation filtering operation, with a learned latent dynamics model making future predictions over a state, and a learned update model refining the latent state with later observations. The attention mechanisms learn to introspect and update specific latent variables. An emission model generates a probabilistic occupancy map as a function of the latent state conditioned on a series of observations.

D. Loss Function

To predict pixel-wise occupancy, a focal loss [28] is used to address class imbalance between occupied and unoccupied pixels. Formally, focal loss $\mathcal{L}_f$, is defined for occupancy prediction logit $\hat{o}$ and ground truth binary occupancy o, where σ is the sigmoid activation function and γ is the focal weighting factor,

$$\mathcal{F} = \sigma(\hat{o})o + (1-o)(1-\sigma(\hat{o})), \tag{13}$$

$$\mathcal{L}_f = -\log(\hat{o})(1-\mathcal{F})^\gamma \tag{14}$$

An additional weighting factor α is used to weight positive samples,

$$\alpha_f = \alpha\hat{o} + (1-\alpha)(1-o) \tag{15}$$

and is uniformly averaged over all time-steps N_{time} and occupancy pixels N_{pixel} for final loss $\mathcal{L}$,

$$\mathcal{L} = \frac{1}{N_{time}N_{pixel}} \sum_{N_{time}} \sum_{N_{pixel}} \alpha_f \mathcal{L}_f. \tag{16}$$

E. Uncertainty Calibration

For an imbalanced binary classification task, focal loss parameters γ and α, can be adjusted to balance the false-negative and false-positive rate. For temporal prediction this is likely to be a function of prediction time. Finding the ideal focal parameters at every time step would entail an expensive hyper-parameter search, so we train with a fixed set of parameters, and calibrate the model output instead by applying a simple scaling function to $\hat{o}$, increasing the decay rate of predicted occupancy by a factor β,

$$\hat{o} = \begin{cases} \beta\hat{o} & \text{if } \hat{o} < 0 \\ \hat{o} & \text{else .} \end{cases} \tag{17}$$

F. Online Inference

A key design feature of MotionPerceiver is the real-time streaming-based architecture. To deploy this in an online application, the latent state that aligns with timing of the next anticipated observation is preserved. Retaining a single latent state entails a constant memory requirement of $N_L \times C_L$. The next scene observation update can be applied to the retained latent state, thereby facilitating the seamless resumption of the forecasting process. This is conceptually a more efficient paradigm compared to more common sequence-based methods. While recurrent models theoretically enable streaming, these models perform poorly in motion forecasting [29], or are applied to latents or tokens produced by passing in fixed windows of information. To the best of our knowledge, there are no streaming-based architectures reported in the literature capable of matching fixed sequence models in performance.

If the sensor sample period and the requested forecasting period differ, more than one $TimePropagate$ function can be trained to evolve the latent state at different periods. This more computationally efficient for inference than a least-common-multiple $TimePropagate$ that needs to be applied multiple times to match the desired period.

V. Experimental Results

A. Training Environment

We train and evaluate on Waymo Open Motion Dataset [30] (WOMD), which consists of $104k$ 9 second sequences captured at 10Hz. Each sequence contains data including agent pose, traffic light signals and road topography. Unless otherwise specified, past phase data is sampled at 2Hz. We use the same inertial frame as the WOMD evaluation benchmark. All models are trained with PyTorch [31][3]. We use the ADAMW optimizer with an initial learning rate of $1e^{-3}$ and fixed batch size between $48-64$, depending on GPU memory available. A polynomial learning rate schedule is adopted with decay power of 0.9 and epoch target of 75. For occupancy focal loss (16), $\alpha = 0.75$ and $\gamma = 2$ are used. To generate an occupancy prediction, the latent state is queried with a evenly spaced $80 \times 80m$ grid of $256 \times 256px$, matching the ground truth occupancy mask. To conserve GPU memory, timesteps to query and apply the loss are sparsely sampled between 0 and 6 seconds. Random sampling mitigates spurious prediction artefacts that may arise on unsampled timesteps if only a consistent set is used. Additionally, we detach gradients between each timestep. This not only speeds up training, but also enforces the Markov assumption (7).

[3]Code is available at https://github.com/5had3z/motion-perceiver.

TABLE I

ARCHITECTURE ABLATION ON WAYMO OPEN MOTION VALIDATION SPLIT

Latent Dims (N x C)	Uncertainty Calibration	Input Data			Conv Decode	Two Phase	Soft IoU			AUC	#
		Agents	Signal	Road			+3s	+6s	Mean	Mean	Params
128x256	✓	✓	-	-	-	-	0.443	0.320	0.424	0.677	5.84M
128x256	✓	✓	✓	-	-	-	0.452	0.329	0.433	0.685	6.57M
128x256	✓	✓	-	✓	-	-	0.454	0.328	0.433	0.684	7.50M
128x256	✓	✓	✓	✓	-	-	0.464	0.333	0.442	0.693	7.56M
64x128	✓	✓	✓	✓	✓	-	0.414	0.308	0.400	0.651	**2.27M**
128x256	-	✓	✓	✓	✓	-	0.405	0.265	0.382	0.716	7.66M
128x256	✓	✓	✓	✓	✓	-	0.486	0.351	0.463	0.710	7.66M
128x256	-	✓	✓	✓	✓	✓	0.501	0.334	0.460	**0.778**	10.7M
128x256	✓	✓	✓	✓	✓	✓	**0.563**	**0.400**	**0.524**	0.772	10.7M

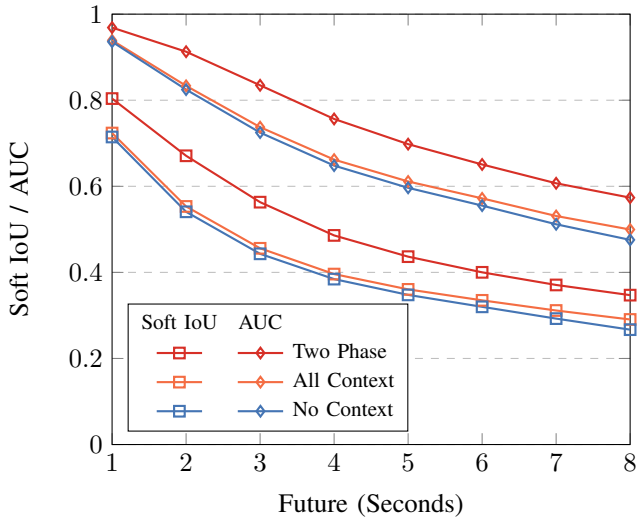


Fig. 4. Soft IoU and AUC at evaluation waypoints on Waymo Open Motion Validation split. Inclusion of contextual features (roadgraph + traffic signals) has a greater effect at later waypoints. Two phase prediction specialization improves performance across the whole sequence.

(a) +0s (b) +2s (c) +4s

Fig. 5. Multi-modal predictions modeled by MotionPerceiver. In this example, the vehicle in the center is predicted to either continue straight or turn left at the intersection. Images are color coded green → true positive (occupancy prediction > 0.5), blue → false positive, red → false negative, black → rasterized road graph.

B. Performance metrics

Performance in occupancy forecasting is characterized using Area Under Curve (AUC) and Soft Intersection-over-Union (Soft IoU). AUC, the area underneath the receiver operating characteristic (ROC) curve, is calculated by sampling the true and false positive rates at positive sample thresholds between 0 and 1. Soft IoU is defined as

$$SoftIoU = \frac{\sum_{N_{pixel}} o\hat{o}}{\sum_{N_{pixel}} (o + \hat{o} - o\hat{o})}. \tag{18}$$

C. General Insights

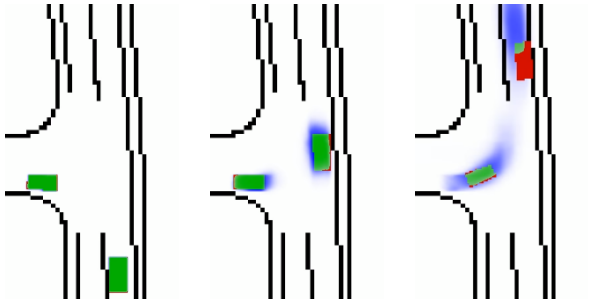
We report architectural and feature ablations of MotionPerceiver in Table I, investigating the effects of latent state dimension sizes, uncertainty calibration and adding scene context. We also explore the effects of "Two Phase" training, discussed further in Section V-D, which uses different $TimePropagate$ modules for the past and future phase. For evaluation, mean Soft IoU and AUC are computed at $1s$ waypoints after the present. In general, MotionPerceiver is conservative at predicting occupancy, overestimating the probability of occupancy when it is unlikely. Hence, applying uncertainty calibration (17) with $\beta = 2$ results in improved Soft IOU and slight degradation in AUC (Uncertainty Calibration, Table I). We also note incorporating contextual information has greater benefit over extended temporal horizons (Fig. 4). Decoding the occupancy query output with a two layer CNN instead of an MLP also shows a modest performance improvement (Conv Decode, Table. I). Hence, a minor trade off between accuracy and latency is available.

We also qualitatively observe several interesting emergent behaviours of the model[4]. $AgentObservation$ performs its function effectively: removing accumulated uncertainty and error in predicted occupancy, and adding unobserved agents to the latent state (Fig. 1). Agents that are unobserved in the update are preserved in the latent state, but their associated occupancy fades over time. For agents in motion, this manifests as a directed smear, attributed to learned uncertainty estimation over the target's future state. Additionally, MotionPerceiver produces multi-modal predictions, particularly at decision boundaries for agents (Fig. 5). MotionPerceiver also shows evidence of learning social dynamics, predicting traffic behaviour such as waiting before merging (Fig. 6).

D. Comparison with Existing Methods

For this evaluation, we introduce two modifications to the general baseline model, sampling all past observations and forecasting at the evaluation period. Hence, two $TimePropagate$ modules are used, we refer to this architecture as "Two Phase". One in the past to propagate

[4]Samples can be viewed in the supplementary material provided at https://sites.google.com/monash.edu/motionperceiver



(a) +0s (b) +2s (c) +4s

Fig. 6. Typical social dynamics are learnt and incorporated in the prediction. In this example, the turning vehicle waits for the vehicle with right of way.

$100ms$ to the next observation, another to propagate at $1s$ increments for forecasting. This simple change improves our benchmark score (Two Phase, Table I) and can indicate that larger $TimePropagate$ increments improves forecasting accuracy, due to less iterations for the same duration. When evaluated on the WOMD withheld test split, our approach demonstrates a significant improvement in Soft IoU compared to existing submissions (Table. II). We note that Soft IoU is a potentially more appropriate statistic for forecasting probabilistic occupancy than AUC, particularly due to the multi-modal nature of the task [32]. Models that are poorly calibrated have significantly worse Soft IoU in Table. II than counterparts with similar AUC, a metric that rewards overconfident unimodal predictions.

TABLE II

WAYMO OPEN MOTION WITHHELD TESTING SPLIT COMPARISON

Model	Soft IoU	AUC	# Params
Spatial Temporal Convolution	0.217	0.744	-
LookAround [11]	0.234	0.801	28.5M
HOPE [4]	0.235	**0.803**	81M
Temporal Query	0.393	0.757	-
Motionnet	0.411	0.694	-
VectorFlow [5]	0.488	0.755	17.1M
STrajNet [8]	0.491	0.778	14.5M
OFMPNet [33]	0.502	0.770	13.3M
YRNet	0.508	0.712	-
Ours	0.523	0.770	**10.7M**
Ours + Occ. Flow	**0.535**	0.779	**10.7M**

E. Multi-task Training

The original WOMD challenge included an "occupancy flow" task to predict where occupancy has originated from, predominately to recover vehicle identities. The task of predicting the past from the current state runs counter to our architecture that intends to predict the future in a uni-directional stream. However, for fair comparison, we jointly train occupancy flow and obtain marginally improved occupancy results (Ours + Occ. Flow, Table II). These marginal gains could potentially be attributed to extra regularization induced by jointly learning a geometrically similar, but distinct task. This is implemented by adding two extra channels to the model output, representing x and y flow, and learned with a simple pixel-wise Huber loss, weighted by 0.1. We note that a simplified version of occupancy flow ground-truth is used, change in vehicle heading is not considered, the change in agent position is simply broadcast over the flow-mask. With this discrepancy in mind, we obtain a flow end-point-error of 4.900 on the test split, in line with other models.

F. Runtime Latency

There is an absence of reporting on the applicability of motion forecasting models for real-time edge inference. Since this is a core benefit of the proposed architecture, we benchmark our system on a common platform, the Nvidia Jetson AGX. To facilitate this, we export MotionPerceiver's modules as individual ONNX models to benchmark with trtexec [5]. We report results in Table III with the following parameters: the number of input tokens for signal and agent updates are 16 and 128 respectively, $OccupancyQuery$ generates a $200 \times 200px$ image, and the rasterized topology input for $RoadgraphEncoder$ is $200 \times 200px$.

TABLE III

TENSORRT INFERENCE LATENCY

Module	Inference (ms)
$StateInitialisation$	1.202
$OccupancyQuery$	9.260
$TimePropagate$	0.905
$AgentObservation$	0.379
$SignalObservation$	0.460
$RoadgraphEncoder$	0.062
$RoadContext$	0.395

Based on these results, our inference latency to forecast $8s$ into the future is $8 \cdot (TimePropagate + RoadContext) = 9.68ms$, assuming that $TimePropagate$ has been trained to predict $1s$ increments. In parallel, another $TimePropagate$ that matches the scene observation period is applied to anticipate the next update.

VI. LIMITATIONS AND FURTHER WORK

MotionPerceiver does not explicitly preserve instance identity in occupancy predictions. This makes the model robust to sensor occlusions and missing information. Although identity is not needed for path planning, future work can explore recovering this by analysing transformer attention.

Importantly, to be used for path-planning, the model needs to be conditioned on ego-agent actions or those of other agents. A method to achieve this could be to inject updates of potential agent trajectories when forecasting. MotionPerceiver is lightweight enough that several trajectories can be proposed in parallel. Future work should focus on integrating this with trajectory optimisation strategies that leverage efficient inference with MotionPerceiver.

VII. CONCLUSION

This paper introduces MotionPerceiver, a motion forecasting architecture designed explicitly for fast and online use. The proposed architecture encodes a scene into a latent state

[5]https://github.com/NVIDIA/TensorRT/tree/main/samples/trtexec

that is evolved forward in time with a learned time evolution function and updated with future observations. This learned recursive state estimation approach is more computationally efficient than existing sequence-based architectures, and performs on-par with larger state-of-the-art models in AUC, and outperforms all others in Soft IoU. Visualized sequences show that MotionPerceiver is able to learn and incorporate typical social dynamics for prediction such as right-of-way, incorporate observations from multiple sources (eg. traffic lights, road graph), and learn to make realistic probabilistic occupancy predictions.

ACKNOWLEDGMENT

This work was supported by an Australian Government Research Training Program (RTP) Scholarship.

REFERENCES

[1] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, O. J. Henaff, M. Botvinick, A. Zisserman, O. Vinyals, and J. Carreira, "Perceiver IO: A general architecture for structured inputs & outputs," in *Int. Conf. on Learning Representations*, 2022.

[2] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," in *Proc. Robotics: Science and Systems*, June 2019.

[3] J. Gu, C. Sun, and H. Zhao, "Densetnt: End-to-end trajectory prediction from dense goal sets," in *Proc. IEEE/CVF Int. Conf. on Computer Vision*, 2021, pp. 15 303–15 312.

[4] Y. Hu, W. Shao, B. Jiang, J. Chen, S. Chai, Z. Yang, J. Qian, H. Zhou, and Q. Liu, "Hope: Hierarchical spatial-temporal network for occupancy flow prediction," *arXiv:2206.10118*, 2022.

[5] X. Huang, X. Tian, J. Gu, Q. Sun, and H. Zhao, "Vectorflow: Combining images and vectors for traffic occupancy and flow prediction," *arXiv:2208.04530*, 2022.

[6] J. Kim, R. Mahjourian, S. Ettinger, M. Bansal, B. White, B. Sapp, and D. Anguelov, "Stopnet: Scalable trajectory and occupancy prediction for urban autonomous driving," in *2022 Int. Conf. on Robotics and Automation (ICRA)*, 2022, pp. 8957–8963.

[7] S. Konev, K. Brodt, and A. Sanakoyeu, "Motioncnn: A strong baseline for motion prediction in autonomous driving," *arXiv:2206.02163*, 2022.

[8] H. Liu, Z. Huang, and C. Lv, "Multi-modal hierarchical transformer for occupancy flow field prediction in autonomous driving," in *2023 IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1449–1455.

[9] R. Mahjourian, J. Kim, Y. Chai, M. Tan, B. Sapp, and D. Anguelov, "Occupancy flow fields for motion forecasting in autonomous driving," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5639–5646, 4 2022.

[10] J. Ngiam, V. Vasudevan, B. Caine, Z. Zhang, H.-T. L. Chiang, J. Ling, R. Roelofs, A. Bewley, C. Liu, A. Venugopal, D. J. Weiss, B. Sapp, Z. Chen, and J. Shlens, "Scene transformer: A unified architecture for predicting future trajectories of multiple agents," in *Int. Conf. on Learning Representations*, 2022.

[11] D. Poplavskiy, "Waymo open dataset occupancy and flow prediction challenge solution: Look around," 2022. [Online]. Available: https://storage.googleapis.com/waymo-uploads/files/research/OccupancyFlow/Dmytro1.pdf

[12] A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, H. Rezatofighi, and S. Savarese, "Sophie: An attentive gan for predicting paths compliant to social and physical constraints," in *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 1349–1358.

[13] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data," in *Euro. Conf. on Computer Vision*. Springer, 2020, pp. 683–700.

[14] B. Varadarajan, A. Hefny, A. Srivastava, K. S. Refaat, N. Nayakanti, A. Cornman, K. Chen, B. Douillard, C. P. Lam, D. Anguelov, and B. Sapp, "Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction," in *2022 Int. Conf. on Robotics and Automation (ICRA)*, 2022, pp. 7814–7821.

[15] Y. Yuan, X. Weng, Y. Ou, and K. Kitani, "Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting," in *Proc. IEEE/CVF Int. Conf. on Computer Vision*, 2021.

[16] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 12 689–12 697.

[17] M. Wang, X. Zhu, C. Yu, W. Li, Y. Ma, R. Jin, X. Ren, D. Ren, M. Wang, and W. Yang, "Ganet: Goal area network for motion forecasting," *arXiv:2209.09723*, 2022.

[18] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," in *Proc. Conf. on Robot Learning*, ser. Proc. Machine Learning Research, vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 86–99.

[19] T. Khurana, P. Hu, D. Held, and D. Ramanan, "Point cloud forecasting as a proxy for 4d occupancy forecasting," *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, pp. 1116–1124, 2023.

[20] X. Liu, M. Gong, Q. Fang, H. Xie, Y. Li, H. Zhao, and C. Feng, "Lidar-based 4d occupancy completion and forecasting," *arXiv:2310.11239*, 2023.

[21] T. Khurana, P. Hu, A. Dave, J. Ziglar, D. Held, and D. Ramanan, "Differentiable raycasting for self-supervised occupancy forecasting," in *European Conf. on Computer Vision*. Springer, 2022, pp. 353–369.

[22] P.-C. Kung, C.-C. Wang, and W.-C. Lin, "Radar occupancy prediction with lidar supervision while preserving long-range sensing and penetrating capabilities," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2637–2643, 2022.

[23] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, "Learning lane graph representations for motion forecasting," in *Euro. Conf. on Computer Vision*. Springer, 2020, pp. 541–556.

[24] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "Vectornet: Encoding hd maps and agent dynamics from vectorized representation," in *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2020, pp. 11 522–11 530.

[25] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, D. Ramanan, P. Carr, and J. Hays, "Argoverse 2: Next generation datasets for self-driving perception and forecasting," in *Proc. Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[26] W. Zhan, L. Sun, D. Wang, H. Shi, A. Clausse, M. Naumann, J. Kümmerle, H. Königshof, C. Stiller, A. de La Fortelle, and M. Tomizuka, "INTERACTION Dataset: An Int., Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps," *arXiv:1910.03088*, Sept. 2019.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[28] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020.

[29] X. Jia, L. Chen, P. Wu, J. Zeng, J. Yan, H. Li, and Y. Qiao, "Towards capturing the temporal dynamics for trajectory prediction: a coarse-to-fine approach," in *6th Annual Conf. on Robot Learning*, 2022.

[30] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov, "Large scale interactive motion forecasting for autonomous driving : The waymo open motion dataset," in *2021 IEEE/CVF Int. Conf. on Computer Vision (ICCV)*, 2021, pp. 9690–9699.

[31] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[32] J. Lobo, A. Jiménez-Valverde, and R. Real, "Auc: A misleading measure of the performance of predictive distribution models," *Journal of Global Ecology and Biogeography*, vol. 17, pp. 145–151, 01 2008.

[33] Y. Murhij, "Ofmpnet: Deep end-to-end model for occupancy and flow prediction in urban environment," 2023. [Online]. Available: https://github.com/YoushaaMurhij/OFMPNet