PORT SAID ENGINEERING RESEARCH JOURNAL
Faculty of Engineering - Port Said University
Volume 29 No. 2 pp: 71:79

Behavior Encoder Transformer BETR: A Transformer Encoder for Predicting Agent Behavior

*Mahmoud Elhusseni [1 *], Yara Maher [2], Emad Elsayed [3], Mohamed F. Abdelkader [4]*

[1] *Communication and Electronics Department, Faculty of Engineering, Port Said University, Port Said, Egypt, email: mahmoud.a.elhusseni@gmail.com*
[2] *Communication and Electronics Department, Faculty of Engineering, Port Said University, Port Said, Egypt, email: yaramaher368@gmail.com*
[3] *Computer and Control Department, Faculty of Engineering, Port Said University, Port Said, Egypt, email: emad.elsayed@eng.psu.edu.eg*
[4] *Communication and Electronics Department, Faculty of Engineering, Port Said University, Port Said, Egypt, email: mdfarouk@eng.psu.edu.eg*

**Corresponding author,* *DOI: 10.21608/pserj.2025.338938.1381*

Received 25-11-2024
Revised 21-12-2024
Accepted 1-1-2025

ABSTRACT

Anticipating the future behavior of road users stands as one of the most formidable challenges in the realm of autonomous driving. Achieving a comprehensive understanding of the dynamic driving environment requires an autonomous vehicle to accurately predict the motion of other traffic participants within the scene. As the complexity of motion prediction tasks increases, capturing intricate spatial relationships, temporal dependencies, and nuanced interactions between agents and map elements becomes crucial. Our proposed hierarchical architecture strategically incorporates transformers, effectively modeling both local and global representations to extract multiscale features. Leveraging the potency of transformers, BETR adeptly captures and encodes intricate patterns of agent interactions, spatial dependencies, and temporal dynamics. Demonstrating superior performance in predicting agent behavior compared to conventional methods, BETR proves its efficacy through extensive experiments. Its capacity to adapt to diverse scenarios establishes BETR as a robust and versatile solution for the intricate task of agent behavior prediction.

Keywords: Transformers, Encoders, Motion Prediction, Vector Representations.

1 INTRODUCTION

Predicting the motion of agents, such as vehicles, pedestrians, and other entities in dynamic environments, holds paramount significance across various domains, ranging from autonomous driving to crowd management and robotics. The ability to forecast the future trajectories of agents facilitates proactive decision-making, enhances safety measures, and optimizes resource allocation in numerous real-world scenarios.

To comprehend the driving environment effectively, an Autonomous Vehicle requires insight into the intentions and future trajectories of all dynamic agents. This capability empowers the Autonomous Vehicle to strategize its trajectory and navigate around obstacles, whether they are stationary (detectable through Object Detection algorithms) or in motion (predicted via Motion Prediction Algorithms).

Therefore, to optimize performance, three primary factors merit consideration: 1) performance metrics, 2) memory usage, and 3) inference time. Addressing these aspects ensures sufficient performance while accommodating deployment on the constrained computational resources of Autonomous Vehicles. To effectively predict an agent's target destination and attain optimal performance according to key metrics in this task, various methodologies have been employed. Some of these approaches involve predicting the agent's final goal and subsequently determining the trajectory from its current position to each predicted destination. The primary objective of such approaches is to enhance accuracy and overall performance, often by leveraging advanced decoder networks to refine predictions.

Recent advancements in motion prediction predominantly utilize either Convolutional Neural Networks (CNNs) [1] or Graph-based message passing techniques [2], [3], [4], [5], [6] to extract local information from different sources, However, CNN-

based approaches face challenges in effectively learning kernels that can adequately cover all scenarios present in the dataset. Consequently, CNN models may struggle to generalize well across diverse scenarios. On the other hand, in graph-based algorithms, all features are typically accorded equal weight. This uniform treatment of features means that outliers and rapid maneuvers exert significant influence on the model's decision-making process, rendering the model susceptible to overfitting due to its complexity.

Despite significant advancements in trajectory prediction, existing methods have notable limitations. One key drawback is that none of these approaches employ a fully transformer-based architecture for predictions, which limits their ability to fully capture complex spatial-temporal dependencies and interactions in dynamic environments. Transformers have proven to be highly effective in various domains, yet their potential remains underexplored in trajectory prediction tasks. Additionally, current methods often fail to effectively utilize available information, such as detailed road layouts and data about adjacent objects. These elements are critical for understanding the driving context, yet conventional models either underutilize them or treat them in isolation, leading to suboptimal performance. Addressing these shortcomings is essential for improving prediction accuracy and robustness in autonomous systems.

In our approach, we focus mainly on enhancing feature extraction of information used as input to our model. Instead of delving into complex decoding processes to attain precise trajectories, we concentrate on refining the feature extraction stage. We employ transformer- based encoders to better extract features as our data set comprises multiple sources of information. Each input scenario may require different sources for decision-making compared to others, leveraging transformer-based encoders provides our model with a significant advantage. It enables the model to learn how to prioritize the most important information sources and allocate greater attention to them. Additionally, it allows the model to disregard misleading information by assigning them lower attention weights. To accomplish this, we utilize self-attention mechanisms within the transformers to determine attention weights dynamically.

An illustrative diagram of our approach is shown in Figure 1.

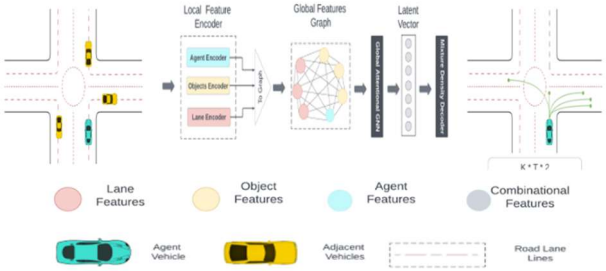

Figure 1: Overview of BETR

To ensure a fair comparison, we standardized all other configurations, including the decoder architecture, where we employed a simple decoder design. Additionally, we utilized the same dataset, Argoverse2 [7], cost function, and all training configurations across different approaches. By adopting benchmark metrics, we facilitated a comprehensive comparison of the various methodologies.

2 LITERATURE REVIEW

In recent times, there has been significant research on motion prediction, driven by the increasing focus on autonomous driving. This prediction commonly relies on input data such as road maps and the historical state of agents. Scene representation methods in this context are generally classified into two categories: rasterized representation and vectorized representation.

2.1 Rasterized Representation

Rasterized encoding methods involve converting high-definition map elements and agents into an image. Early works [8], [9], [10], [11], [12], [13], [14] utilized Convolutional Neural Networks (CNNs) to encode the image. However, these methods face limitations in capturing the structural information present in high-definition maps. Multipath [15], employed CNNs to extract features from raster images, enabling the prediction of probabilities for K predefined anchor trajectories and the regression of offsets from the anchor states.

2.2 Vectorized Representation

Vectorization involves the conversion of data into numerical format represented by vectors. VectorNet [2] stands out as the first approach to directly integrate vectorized information for both lanes and agents. It has gained widespread adoption in recent research efforts [3], [16], [5], [17] due to its efficiency and scalability. In VectorNet, road maps and agent trajectories are represented as polylines.

In the proposed work we utilized Goal-based Trajectory Prediction. This approach frames the prediction task as a planning problem by introducing the pedestrian's goal as a latent variable. In the TNT method

[18], anchors are first sampled from road maps, and trajectories are generated based on these anchors. Similarly, LaneRCNN [19] uses a decoding pipeline where lane segments serve as anchors. Each anchor is assigned a probability, and non-maximum suppression (NMS) is applied to remove duplicate goals in close proximity. DROGON [20], on the other hand, focuses on intentional destinations for individual agents. It introduces a trajectory prediction dataset to analyze goal-oriented behavior and employs a conditional VAE framework to predict multiple plausible trajectories. Goal-based modeling has also been used in optimizing planning policies for autonomous driving [21] and predicting human trajectories [22]. Unlike these methods, BETR takes an anchor-free approach to goal-based modeling, enabling it to be trained in an end-to-end manner.

3 The Proposed Behavior Encoder Transformer (BETR)

The proposed BETR (Behavior Encoder Transformer) method is an end-to-end trajectory prediction approach utilizing transformer-based architecture to forecast the complete future trajectory of an agent. In this method, each future point is treated as an individual prediction task independent of other predicted time steps. We first use a vectorized representation [2] to represent our source of information in each scene. We use a transformer based local encoder to extract features from each source of information separately, then combine all vectors from different sources together into a global feature encoder. A simple decoder network takes the latent vector output of the global feature encoder and predicts the agent's behavior.

3.1 Sources of Information

Preprocessing our data stands out as one of the pivotal steps in this task. The input data provided to the model during both training and evaluation is a critical indicator of the model's performance. Our approach to preprocessing closely aligns with the methodology proposed by VectorNet [2]. Specifically, we categorize the information describing each scene into three primary sources: Agent-related information, Adjacent Objects-related information, and Lane Lines-related information.

Utilizing a vectorized representation, we segment all sources into vectors in the spatial domain for L lane lines and the temporal domain for A, O agent and adjacent objects data.

The vectors for agent data are structured as follows:

$$a_i = \{xs, ys, xe, ye, vx, vy, h, cd\}$$

where:

• (*xs, ys*): Represents the initial (x, y) coordinates of vector *i* relative to the agent's current position.
• (*xe, ye*): Denotes the final (*x, y*) coordinates of vector is related to the agent's current position.
• (*vx, vy*): Signifies the average velocity in the *x* and *y* directions during the temporal interval of the vector.
• *h*: Indicates the average heading (yaw angle) of the agent during the temporal interval of the vector.
• *cd*: Represents the count of dynamic objects within a predefined threshold surrounding the agent.

For object data, the vectors are represented as:

$$o_i = \{xs, ys, xe, ye, ts, D_a, \theta_a, vx, vy, h, \text{object type}\}$$

where:

• *ts*: indicates timestamp of this vector.
• *Da*: Represents the average distance from the agent during the vector interval.
• θa : Signifies the average angle between the objects and the agent during the vector interval.
• Object type: Specifies the type of object represented by the vector.

Lastly, lane data is organized into vectors denoted by:

$$l_i = \{xs, ys, zs, xe, ye, ze, \text{Intersection}, \theta_d, \text{type}\}$$

where:

• (*xs, ys, zs*): Depicts the starting (*x, y, z*) coordinates of vector i for this lane line relative to the agent's current position.
• (*xe, ye, ze*): Represents the ending (*x, y, z*) coordinates of vector *i* for this lane line relative to the agent's current position.
• *I*(Intersection): binary flag indicates whether the lane line of this vector is an intersection or not.
• *θd* : Denotes the angle of the vector.
• *lanetype*: Specifies the type of lane represented by the vector.

This preprocessing stage ensures that the data is appropriately formatted and ready for input into the model.

3.2 Local Feature Encoder

After vectorizing our information sources, the vectors are processed by individual local encoders, each tailored to a specific source. These encoders are built upon the Transformer architecture, as outlined in [23], utilizing a self-attention mechanism to effectively encode local vectors. The model combines vectors of certain elements with varying weights, reflecting their importance to the scene. The importance of a vector in a scene is computed according to the projection of its features. We stack L layers of transformer blocks, allowing vectors to pass from one layer to another, thus enriching them with more information about other vectors. For agent vectors:

$$\alpha_{ij} = a_i^l \cdot W_q^l \times \left(a_j^l \cdot W_k^l\right)^T \quad (1)$$

$$v_i^l = \sum_{j=0}^{N_a} a_{ij} \times a_j \cdot W_v^l \quad (2)$$

$$a_i^{l+1} = v_i^l + \pi\left(v_i^l\right) \quad (3)$$

Where α_{ij} is the attention weight between vector a_i and vector a_j, W_q, W_k, W_v are learnable parameters, and $\pi(.)$ is a learnable Multi-Layer Perceptron (MLP) with two linear layers and a ReLU activation function between them:

$$\pi(.) = (W_2^l)^T \max\left(0, (W_1^l)^T \cdot \left(v_i^l\right)\right) \quad (4)$$

Here, $a_i^1, o_i^1, l_i^1 \in R^{d_a}$ and $a_i^l, o_i^l, l_i^l \in R^{d_h}$ and $W_q, W_k, W_v \in R^{d_i \times d_h}$.

After propagation through L layers, we obtain three matrices of vectors corresponding to agents, objects, and lanes, each offering an enhanced representation of the input scene.

3.3 Global Vectors Interaction

Using vector matrices outlined above, we construct a fully connected graph where each vector corresponds to a node within the graph, as illustrated in Figure 1. In this graph, every node is interconnected with all others. Subsequently, we feed this graph into a Graph Neural Network (GNN) to encode global features encompassing all sources of information. We employ the global soft attention technique introduced in [24], where a Global Attentional Graph Neural Network dynamically allocates attention across each node. The attention weights are based on latent vectors representing individual elements within the graph:

$$r_i = \sum_{n=1}^{N_i} \text{softmax}\left(h_{\text{gate}}(x_n)\right) \cdot h_\theta(x_n) \quad (5)$$

Where $h_{gate}: R^F \rightarrow R$ and h_θ denote MLPs.

The GNN generates a final latent vector, denoted as $h_i \in R^d$, which encapsulates all scene information. This vector serves as a comprehensive representation from which all features relevant to trajectory prediction within the scene are extracted.

3.4 Mixture Density Decoder

Our task is to predict the joint probability distribution for a scene $p(x_0, x_1, \ldots, x_{T-1}, y_0, y_1, \ldots, y_{T-1} \mid A, O, L)$ as illustrated in Figure 2, where T is the number of future frames to predict. Instead of predicting a Gaussian probability distribution for each coordinate and time step pair, we predict Mixture Gaussian Components with Identity variance and with multiple confidences for each mean. Each μ_k represents a single pair $(x \mid y)_t$ in trajectory k. Our goal is to predict K different trajectories with $C \in R^K$ confidence for each trajectory. This approach allows for a more flexible representation of uncertainty in the predicted trajectories.

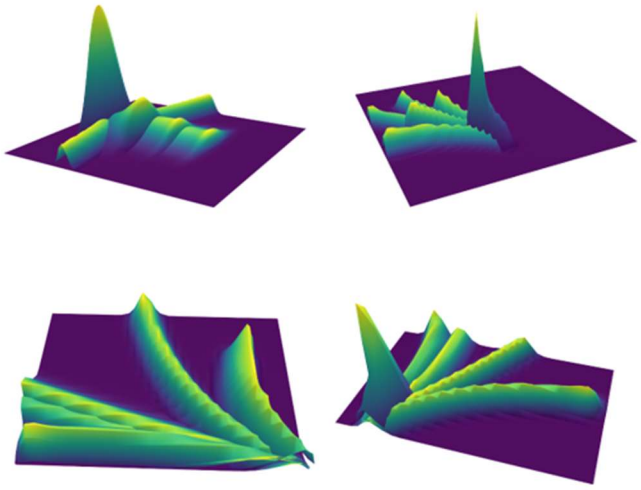

Figure 2: Sample output distribution where the model predicts multiple trajectories, each consisting of multivariate Gaussian distributions. Model Predicts K Trajectories (e.g. 6 trajectories as illustrated in figure), for each trajectory there are T time steps represented by P(x, y) Gaussian distribution.

3.5 Loss Function

In our experiments, we investigated the impact of two approaches to improving model performance. Initially, across all experiments, we trained our model using the negative log likelihood loss function. To enhance our model's effectiveness further, we introduced a regularization strategy in the BETR-Attn penalty experiment.

This involved incorporating an attention penalty during training to regulate the attention mechanism.

1) Negative Log Likelihood: To evaluate the effectiveness of our predictions, we propose a cost function l based on the negative likelihood of the output probability distribution. This cost function serves as a measure of how well our model's predictions align with the observed data, with lower values indicating better performance:

$$l_\theta = -\log\left(p(x0, \ldots, y_{T-1} \mid A, O, L; \theta)\right) \quad (6)$$

$$l_\theta = -\log\left(\sum_{k=1}^{K} e^{\left(\log(c_k) - \frac{1}{2}\sum_{t=1}^{T}\left(x_{gt,t} - x_{k,t}\right)^2 - \left(y_{gt,t} - y_{k,t}\right)^2\right)}\right) \quad (7)$$

2) Attention Penalty: Our model primarily focused on the agent and often neglected critical information about lanes and objects in the scene. This limitation hindered the model's ability to comprehensively understand the environment and make informed decisions. To address this issue, we introduced the attention penalty to enforce the model to focus across all relevant information within the scene. The penalty aimed to enhance the model's capacity to capture essential details during training. This regularization technique was instrumental in discouraging overly diffuse or unfocused attention patterns, ultimately leading to improved performance.

Through this enhancement, our model became more adept at processing complex scenes and making informed decisions based on a comprehensive understanding of its surroundings.

$$penalty = \lambda \frac{1}{N} \sum_{i=1}^{N} (attention_i - target_i)^2 \quad (8)$$

Where:

- $target_i = \frac{1}{SeqLen}$
- $L_{total} = L_{nll} + Penalty$

3.6 Handling Variable Number of Objects

In the proposal discussed, a challenge arose from the variable number of adjacent objects and lane lines present in each scene. Consequently, each local encoder has to handle variable batch sizes during forward and backward propagation for each scene. To mitigate this issue, we endeavored to maintain a constant number of objects and lanes in each scene. This adjustment aimed to enable the utilization of transformers in local encoders without instability arising from the variable input size across scenes.

1) Padding Objects and Lanes: To ensure efficient utilization of transformers, we opted to pad the objects and lanes in all scenes to a fixed number. This approach allows for consistent processing across scenes. When padding objects and lanes, it is advisable to incorporate redundant or less crucial elements. This strategy encourages transformers to allocate less attention or disregard these padded elements during the decision-making process.

Padded Objects vectors: We chose to pad objects with distant stationary objects from the remote past, thus defining:

- $xs, ys, xe, ye = (\infty, \infty, \infty, \infty)$
- $Ts = -\infty$
- $Da, \theta_a = (\infty, 0)$
- $vx, vy = (0,0)$
- Object type = 0

$$oi = [\infty, \infty, \infty, \infty, -\infty, \infty, 0.0, 0.0, 0.0, 0.0, 0.0]$$

We chose to pad lanes with distant lanes, thus defining:

- $xs, ys, zs, xe, ye, ze = (\infty, \infty, \infty, \infty, \infty, \infty)$
- I(Intersection), $\theta_d = (0,0)$
- $Da, \theta_a = (\infty, 0)$
- $vx, vy = (0,0)$
- Lane type = 0

$$l_i = [\infty, \infty, \infty, \infty, \infty, \infty, 0, 0.0.0, 0]$$

2) Averaging Objects and Lane: Another approach utilized involves aggregating all object vectors and lane vectors into two distinct groups. This aggregation simplifies the representation of objects and lanes, enhancing computational efficiency and reducing complexity.

4 EXPERIMENTS

4.1 Experimental Setup

Dataset. BETR is trained on Argoverse2 [7]. The motion forecasting dataset is a curated collection comprising three datasets tailored for perception and forecasting research within the self-driving domain. Among these datasets, the annotated Sensor Dataset stands out. This comprehensive dataset encompasses high-resolution imagery captured by seven ring cameras and two stereo cameras, in addition to LIDAR point clouds and 6-DOF map-aligned pose information. Each sequence within the dataset is annotated with 3D cuboid annotations for 26 object categories, ensuring thorough sampling to facilitate the training and evaluation of 3D perception models. The dataset comprises a total of 303,039 real-world driving scenarios, which have been divided into distinct training, validation, and test sets. Specifically, the training set consists of 199,908 scenarios, while the validation set comprises 24,988 scenarios. Additionally, the test set contains 78,143 scenarios, ensuring a comprehensive and balanced distribution across the dataset subsets for robust evaluation and training purposes. In both the training and validation sets, all scenarios are represented as 11-second sequences sampled at 10 Hz. Each sequence is divided into two segments: the first 6 seconds consist of past tracking data, while the subsequent 5-second segment contains ground truth information. However, it's important to note that only the first 6-second trajectories are publicly available in the test set, implying that the ground truth information for the final 5 seconds is not included in the test set.

Training details. In our training process, which spanned over 10 days, we trained the model for 100 epochs utilizing a single NVIDIA RTX A4000 GPU. We employed a learning rate of 0.001 and implemented a cosine learning rate scheduler. It's important to note that we did not utilize any data augmentation techniques during the training phase.

BETR Implementation. We began by offline transforming the Argoverse2 raw dataset into an intermediate format, which consists of matrices of vectors representing each source of information. These transformed data were stored for subsequent use.

During forward propagation, the data are loaded for each source separately and encoded using Transformer-based Local Encoders. Specifically, agent data per batch is represented by a matrix $A \in R^{bs \times 50 \times 8}$, adjacent objects are represented by a matrix $O \in R^{\sum_{o=1}^{O} o \times 60 \times 11}$, and road lane lines are represented by a matrix $L \in R^{\sum_{l=1}^{NL} l \times 60 \times 9}$. After each Local Encoder layer block, each block of Agent Encoder, Object Encoder, Lane Encoder produces matrices$\hat{A} \in R^{bs \times 16}$, $\hat{O} \in R^{bs \times No \times 16}$, $\hat{L} \in R^{bs \times Nl \times 16}$

where for training bs=64 and for evaluation bs=128 respectively. Subsequently, the Global Feature Encoder (Global Vectors Interaction Layer) combines these three matrices into a number of fully connected graphs, one for each scene. Each graph is represented by a data matrix $D \in R^{(1+N_o+N_l)\times 16}$ and an adjacency matrix representing all edges between all nodes in the graph. These graphs are then passed through an Attentional Graph Neural Network to generate a latent vector that holds features for the global scene $\widehat{H \in R^{64}}$. We implemented a simple decoder consisting of three Multi-Layer Perceptrons (MLPs) to decode the latent vector extracted by the Encoder Network. The decoder outputs K=6 trajectories represented by two vectors: a Confidence Vector and a predicted Trajectory vector $\hat{y} \in R^{6\times 50\times 2}$

For parameter optimization, we utilized the Negative Likelihood cost function. This function aids in optimizing the parameters of the model during training, ensuring effective learning and prediction performance.

Other models implementation. We introduced certain modifications to ensure comparability with our approach. These modifications were aimed at maintaining consistency across all models' components, while varying only the encoder architecture.

DenseTNT: In the case of Dense TNT, we adjusted by replacing its complex decoder with our standard decoder, consisting of three multi-layer Perceptrons. Furthermore, we replaced the loss function utilized in DenseTNT with the negative log likelihood loss function to align it with our approach.

Metrics. In our evaluation process, we adopt metrics proposed by the Waymo Open Motion Dataset [25]. Specifically, we utilize the following metrics to assess the performance of our model:

1) Minimum Final Displacement Error (minFDE):
This metric measures the minimum Euclidean distance between the predicted final position and the ground truth final position of the agent.

2) Minimum Average Displacement Error (minADE): This metric calculates the minimum average Euclidean distance between the predicted trajectory and the ground truth trajectory over a specified time horizon.

3) Miss Rate (MR): The Miss Rate quantifies the proportion of predictions that fail to capture the ground truth trajectory within a certain threshold. Given that each prediction consists of 50 (x, y) pairs, we calculate the Miss Rate at different time frames, denoted as MR@{10, 30, 50}.

These metrics provide comprehensive insights into the accuracy and reliability of our predictions across various aspects of motion forecasting, enabling thorough evaluation and comparison with benchmark datasets.

4.2 Main Results

We conducted an evaluation of our approach on the Argoverse validation set and presented the results in Table 1. It is evident that BETR outperforms all other models, which are built on the same encoder and other configurations, across most proposed metrics with a remarkable margin. Even when compared to models with more sophisticated decoders and more complex architectures, our approach demonstrates competitive performance and achieves state-of-the-art results.

Table 1. Comparison of performance metrics usage for various encoder approaches.

Encoder	Model	minFDE	avgADE	avgMR	MR@10	MR@30	MR@50
Traditional Techs	Poly-fit	4.55	2.78	-	-	-	-
	Interp1d	2.67	1.25	-	-	-	-
Convolution based	CNN Motion	1.65	1.07	0.02	0.03	0.03	0.00
Graph based	DenseTNT	1.55	1.03	0.02	0.02	0.02	0.00
Transformer based	BETR	1.42	1.06	0.03	0.03	0.03	0.01
	BETR-Norm	1.52	1.00	0.01	0.01	0.01	0.00
	BETR-pad	1.39	0.95	0.02	0.02	0.02	0.00
	BETR-penalty	0.87	1.25	0.01	0.00	0.00	0.00

Table 2. Comparison of performance resource usage for various encoder approaches.

Encoder	Model	Model Size (MB)	CPU Time(s)	GPU Time(ms)
Traditional techs	Poly-fit	-	0.0003	0.344
	Interp1d	-	0.0002	0.248
Convolution based	CNN Motion	158.42	0.146	1.515

Graph based	DenseTNT	0.7156	11.495	2.993
Transformer based	BETR	0.6061	0.8001	0.208
	BETR-Norm	0.6061	0.7989	0.208
	BETR-pad	0.6147	2.0186	0.525
	BETR-penalty	0.6061	0.8001	0.208

These findings underscore the efficacy and robustness of BETR in motion forecasting tasks, highlighting its potential for real-world applications in the self-driving domain. When evaluating model complexity and inference time, prioritizing these metrics becomes essential to facilitate efficient deployment on lightweight microcontrollers and support real-time applications. The data presented in Table 2 underscores the suitability of BETR for such deployments, particularly in contexts like self-driving cars, where speed and efficiency are paramount.

4.3 Ablation Study

Transformer-Based Local Encoder. As shown in Figure 3, Transformers are utilized to capture spatial relationships, temporal dependencies, and interactions between agents and map elements. Our architecture adopts a hierarchical approach, enabling the learning of both local and global representations. This hierarchical strategy proves beneficial for the model, as it facilitates the acquisition of multiscale features while maintaining efficiency.

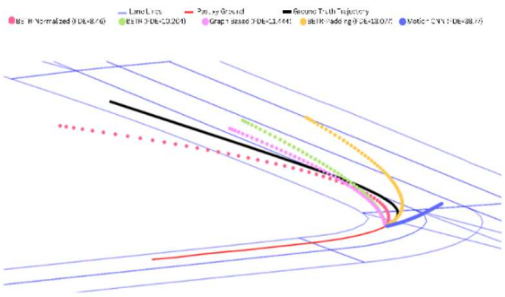

Figure 3: A comparison between different models in this scenario. Black Solid line represents Ground Truth for this scenario, Red Solid Line represents Past trajectory input to the model, CNN Motion [1] model (Blue dotted line), based on CNN feature extractor, could not extract much information about road lane lines, thereby failing to accurately understand correct behavior of the agent.

It appears that using transformers, the model can better capture all sources of information and choose the most important set of input sources to make its decision.

Mixture Density Decoder. The decoder in our model exhibits a multimodal prediction output, as illustrated in Figure 2, where it generates multiple potential trajectories and assigns a confidence score to each trajectory.

Transformer Penalty. Figure 4 illustrates the evaluation of regularization's impact on the model's attention behavior and performance. Prior to integrating the attention penalty, depicted in Figure 4(a), the model predominantly focused on gathering information about the agent while neglecting details about objects and lanes. This limitation is evident in the global attention map of Figure 4(a), where crucial information remains uncaptured, leading to suboptimal decision-making.

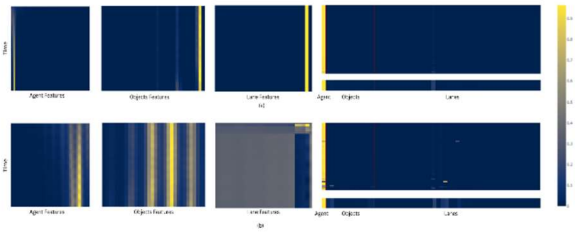

Figure 4: Attention maps generated by each encoder and the global attention map. (a) Before the integration of the attention penalty. (b) After integrating the attention penalty into the model.

However, upon integrating the attention penalty into our model, depicted in Figure 4(b), significant improvements are observed. The attention penalty encourages the model to capture essential details about objects and lanes, in addition to information about the agent. As a result, our model demonstrates improved proficiency in analyzing complex scenes and making well-informed decisions.

5 CONCLUSION

The Behavior Encoder Transformer (BETR) introduced in this paper represents an advancement in the domain of predicting agent behavior. By harnessing the power of transformer encoders, BETR excels in capturing intricate patterns and dependencies within agent trajectories, offering a robust solution for dynamic scenario predictions. The model's ability to efficiently encode and learn representations of complex behaviors at various scales contributes to its superior performance. The empirical evaluation showcased the effectiveness of BETR across diverse datasets, demonstrating its adaptability and generalization capabilities. As we move forward, the insights gained from BETR's success pave the way for further research in refining transformer-based approaches for a broader range of applications in autonomous systems.

REFERENCES

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

[2] 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," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 11 525–11 533.

[3] J. Gu, C. Sun, and H. Zhao, "Densetnt: End-to-end trajectory prediction from dense goal sets," Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 15 303–15 312.

[4] S. Shi, L. Jiang, D. Dai, and B. Schiele, "Motion transformer with global intention localization and local movement refinement," Advances in Neural Information Processing Systems, vol. 35, 2022, pp. 6531–6543.

[5] J. Ngiam, B. Caine, V. Vasudevan, Z. Zhang, H.-T. L. Chiang, J. Ling, R. Roelofs, A. Bewley, C. Liu, A. Venugopal, and D. Weiss, "Scene transformer: A unified architecture for predicting multiple agent trajectories," arXiv preprint, 2021.

[6] L. Zhang, P. Li, J. Chen, and S. Shen, "Trajectory prediction with graph-based dual-scale context fusion," in 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, October 2022, pp. 11 374–11 381.

[7] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, and D. Ramanan, "Argoverse 2: Next generation datasets for self-driving perception and forecasting," arXiv preprint, 2023.

[8] S. Park, G. Lee, J. Seo, M. Bhat, M. Kang, J. Francis, A. Jadhav, P. P. Liang, and L.-P. Morency, "Diverse and admissible trajectory forecasting through multimodal context understanding," in Computer Vision–ECCV 2020, ser. Proceedings, Part XI, vol. 16. Springer International Publishing, 2020, pp. 282–298.

[9] F. Marchetti, F. Becattini, L. Seidenari, and A. D. Bimbo, "Mantra: Memory augmented networks for multiple trajectory prediction," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 7143–7152.

[10] S. Casas, C. Gulino, R. Liao, and R. Urtasun, "Spatially-aware graph neural networks for relational behavior forecasting from sensor data," arXiv preprint, 2019.

[11] N. Djuric, V. Radosavljevic, H. Cui, T. Nguyen, F. Chou, T. Lin, N. Singh, and J. Schneider, "Uncertainty-aware short-term motion prediction of traffic actors for autonomous driving," in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2020, pp. 2095–2104.

[12] Y. Zhang, J. Zhang, J. Zhang, J. Wang, K. Lu, and J. Hong, "A novel learning framework for sampling-based motion planning in autonomous driving," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 01, 2020, pp. 1202–1209.

[13] Y. Biktairov, M. Stebelev, I. Rudenko, O. Shliazhko, and B. Yangel, "Prank: Motion prediction based on ranking," in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 2553–2563.

[14] S. Casas, A. Sadat, and R. Urtasun, "Mp3: A unified model to map, perceive, predict and plan," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 14 403–14 412.

[15] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," arXiv preprint, 2019.

[16] Q. Sun, X. Huang, J. Gu, B. Williams, and H. Zhao, "M2i: From factored marginal trajectory prediction to interactive prediction," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 6543–6552.

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

[18] H. Zhao, J. Gao, T. Lan, C. Sun, B. Sapp, B. Varadarajan, Y. Shen, C. Li, and C. Schmid, "TNT: Target driven trajectory prediction," in Conference on Robot Learning. PMLR, 2021, pp. 895–904.

[19] W. Zeng, M. Liang, R. Liao, and R. Urtasun, "Lanercnn: Distributed representations for graph-centric motion forecasting," arXiv preprint, 2021.

[20] C. Choi, A. Patil, and S. Malla, "Drogon: A causal reasoning framework for future trajectory forecast," arXiv preprint, vol. 2, no. 3, p. 4, 2019.

[21] S. Albrecht, C. Brewitt, J. Wilhelm, B. Gyevnar, F. Eiras, M. Dobre, and S. Ramamoorthy, "Interpretable goal-based prediction and planning for autonomous driving," in 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021, pp. 1043–1049.

[22] H. Tran, V. Le, and T. Tran, "Goal-driven long-term trajectory prediction," in Proceedings of the IEEE/CVF

Winter Conference on Applications of Computer Vision, 2021, pp. 796–805.
[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Kaiser, and I. Polosukhin, “Attention is all you need,” Advances in Neural Information Processing Systems, vol. 30, 2017, pp. 5998–6008.

[24] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” arXiv preprint, 2015.

[25] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. Qi, Y. Zhou, and Z. Yang, “Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 9710–9719.