Deep Generative Models Project

Benchmark Framework for Computational Efficiency in Trajectory Prediction

Ali Ekhterachian, Matin M.Babaei, Amin Mirzaei

Supervised by: Dr.Bahari

October 20, 2025

Abstract

Trajectory prediction is a cornerstone of applications such as autonomous driving, robotics, and surveillance, where timely and precise forecasting of movement patterns is critical. Recent evaluation paradigms predominantly emphasize predictive accuracy, often at the expense of computational efficiency—a decisive factor for real-time systems operating under resource constraints. This paper seeks to establish a comprehensive benchmarking framework to systematically evaluate trajectory prediction models with an explicit focus on computational efficiency. By integrating metrics for memory usage, inference latency and energy consumption, this framework aims to provide actionable insights for designing models optimized for both accuracy and efficiency. The deliverables include an open-source benchmarking framework and a dataset of evaluated results, which will facilitate the adoption of sustainable and scalable predictive models in resource-constrained environments.

1 Introduction

Trajectory prediction underpins numerous real-world applications where anticipating the movement of objects or agents is essential. Autonomous vehicles, for instance, must predict the trajectories of pedestrians, cyclists, and other vehicles to navigate safely and efficiently. Similarly, robots engaged in human-robot interactions or surveillance systems analyzing crowded environments depend on accurate and real-time predictions. Despite advancements in predictive accuracy, the computational complexity of state-of-the-art models presents significant challenges for deployment in latency-critical scenarios. Real-world systems often operate under strict computational budgets, where inefficiencies can lead to delayed responses, reduced performance, or outright system failures.

Recent benchmarking practices in trajectory prediction research overwhelmingly prioritize accuracy metrics such as Average Displacement Error (ADE) and Final Displacement Error (FDE). While these metrics are indispensable for assessing model precision, they offer little insight into the computational feasibility of deploying these models in constrained environments. As the demand for scalable, efficient, and environmentally sustainable AI systems grows, there is a pressing need for evaluation frameworks that balance accuracy with computational efficiency. Addressing this gap will not only improve the deployability of trajectory prediction models but also promote a holistic understanding of their performance characteristics.

The lack of standardized tools and methodologies for evaluating computational efficiency in trajectory prediction models has created a significant barrier to progress in the field. Key challenges include:

1. **Absence of Comprehensive Metrics:** There is no universally accepted set of metrics for systematically assessing computational performance.
2. **Limited Cross-Model Comparability:** Researchers lack standardized benchmarks to enable meaningful comparisons of models across diverse hardware platforms and deployment contexts.
3. **Unexplored Trade-offs:** There is insufficient understanding of how computational efficiency interacts with predictive accuracy, leaving practitioners without clear guidelines for balancing these competing priorities.

2 Related Works

2.1 Frameworks and Toolkits for Benchmarking Computational Efficiency in AI and Real-Time Systems

2.1.1 RT-Bench: an Extensible Benchmark Framework for the Analysis and Management of Real-Time Applications (2022)

RT-Bench [1] is a benchmarking framework focused on real-time systems, specifically evaluating the latency, scalability, and predictability of workloads under time-critical constraints. It is designed to measure the performance of tasks that must operate within strict timing deadlines, making it relevant for embedded systems, robotics, and other real-time applications. RT-Bench emphasizes system-level behavior, including factors like response time under variable workloads and resource contention, offering insights into how well systems handle real-time demands.

2.1.2 MLPerf Inference Benchmark

MLPerf is a comprehensive benchmarking suite for evaluating the performance of machine learning models across a variety of tasks, including image classification, object detection, and natural language processing. It focuses on both training and **inference** performance, providing metrics like throughput, latency, and accuracy. We only refer to MLPerf Inference Benchmark [2] here. MLPerf supports a wide range of hardware platforms (e.g., GPUs, TPUs, edge devices) and frameworks (e.g., TensorFlow, PyTorch). Its goal is to standardize ML evaluation, encourage optimization across hardware and software stacks, and support community-driven improvements in AI system performance.

2.1.3 Batch Prompting: Efficient Inference with LLM APIs [3]

In the realm of large language models (LLMs), inference on extensive datasets can be both computationally and financially demanding. Cheng et al. (2023) introduced Batch Prompting, a method that enables LLMs to process multiple samples simultaneously by batching prompts. This approach significantly reduces token and time costs while maintaining or even enhancing performance across various tasks, including commonsense question answering, arithmetic reasoning, and natural language inference. The authors demonstrated that, under a few-shot in-context learning setting, inference costs decrease almost inverse linearly with the number of samples in each batch. Empirical evaluations showed that Batch Prompting could achieve up to a $5\times$ reduction in inference costs with six samples per batch, without compromising accuracy.

2.1.4 UniTraj [4]

UniTraj is a unified benchmark framework designed to assess both accuracy and efficiency in trajectory forecasting models. It provides standardized datasets, predefined evaluation metrics, and a fair comparison protocol to measure inference speed and resource consumption. By incorporating computational efficiency as a core evaluation criterion, UniTraj enables researchers to analyze the trade-offs between model complexity and performance.

3 Framework Methodology

3.1 Metric Design

Trajectory prediction research has predominantly focused on accuracy-oriented benchmarks. The two most widely cited metrics are:

- **Average Displacement Error (ADE):** Measures the average Euclidean distance between predicted trajectories and ground truth trajectories over all time steps.
- **Final Displacement Error (FDE):** Focuses on the Euclidean distance at the final predicted time step, highlighting endpoint accuracy.

Other supplementary metrics include:

- **Miss Rate (MR):** Measures whether the predicted trajectory stays within a defined margin around the ground truth.
- **Collision Rate:** Evaluates safety in multi-agent environments by checking for collisions between predicted paths.

While these metrics provide valuable insights into predictive performance, they focus exclusively on trajectory accuracy and safety, often overlooking how computational resources are consumed during inference. We need to introduce new metrics and our selected metrics are:

1. **Memory Consumption:** Quantifying peak memory usage during model inference to identify resource-intensive processes.

2. **Inference Latency:** Measuring the time required for a single prediction across varying input sizes and hardware configurations.

3. **Scalability:** Evaluating the model's ability to maintain performance under increasing workloads or on resource-constrained hardware.

4. **Energy Usage:** Assessing power consumption during inference to promote energy-efficient model design.

3.2 Selected Models, Hardware and Datasets

For benchmarking, we selected four state-of-the-art trajectory baseline prediction models:

- AutoBots [5]: AutoBot is a transformer-based model that achieves competitive results across multiple trajectory prediction benchmarks. It leverages equivariant feature learning to model the joint distribution of trajectories using multi-head attention mechanisms, enabling effective multi-agent interaction modeling.

- MTR [6]: MTR was the top-performing model in the WOMD Challenge 2022. It integrates global intention priors with local motion refinement, employing a small set of adaptable motion query pairs. This approach allows for accurate trajectory predictions while effectively capturing different motion types.

- EqMotion [7]: EqMotion is a recent transformer-based model designed for trajectory prediction. It introduces an equivariant motion representation, ensuring consistency across transformations such as rotation and translation. By leveraging self-attention mechanisms and structured feature learning, EqMotion effectively captures spatial-temporal dependencies in motion data.

Inference was conducted on two hardware platforms:

1. NVIDIA Quadro RTX 5000 (high-performance GPU for workstation-level evaluation)

2. NVIDIA Jetson Nano Devkit 4Gb (low-power embedded device for edge computing scenarios)

We chose the NVIDIA Jetson Nano Devkit (4GB) as one of the evaluation platforms because it represents a low-power, edge computing environment, which is critical for real-world deployment scenarios where computational resources are limited. Many real-time trajectory prediction applications, such as autonomous drones, robotics, and IoT-based surveillance, require models to run efficiently on embedded systems with constrained resources. Evaluating models on Jetson Nano helps assess their viability for such use cases. Moreover, Comparing model performance across a high-end GPU (RTX 5000) and an embedded GPU (Jetson Nano) highlights how well different models scale across hardware architectures.

The datasets used in this evaluation consist of trajectory data, primarily represented as sequences of spatial coordinates (x, y) over time. These datasets are widely used in motion prediction tasks, providing real-world scenarios where agents such as pedestrians or vehicles move in dynamic environments. They are briefly discussed below:

1. ETH/UCY: This dataset consists of pedestrian trajectories collected from real-world crowded urban spaces. The data is typically provided as (x, y) coordinates of pedestrians over time, captured at fixed intervals. It is commonly used for studying human motion prediction in social settings.

2. TrajNet++: A large-scale benchmark dataset containing various human and vehicle trajectories in diverse environments. It provides sequences of (x, y) coordinates along with scene context. The dataset is designed to evaluate models in complex interaction scenarios.

3. nuScenes: A dataset specifically created for autonomous driving applications. It includes vehicle trajectories represented as (x, y) coordinates, along with rich multimodal sensor data such as LiDAR point clouds, camera images, and radar data. However, in this study, we focus on the trajectory data.

4. Waymo: A large-scale self-driving dataset that includes detailed vehicle trajectories captured from real-world driving scenarios. Similar to nuScenes, it provides (x, y) coordinates of vehicles over time, along with extensive sensor data (LiDAR, cameras, etc.).

3.3 Metric Evaluation

To assess computational efficiency, we measured the following key performance metrics:

1. Inference Time: We recorded the average inference latency over multiple runs to ensure statistical robustness.

2. Memory Usage: GPU memory consumption was monitored using torch.cuda.memory_allocated() to track model memory footprint during inference.

3. Scalability: We evaluated the impact of increasing batch sizes on inference time and memory usage to understand how well each model scales with workload intensity.

4. Energy Consumption: On Jetson Nano, we measured power draw using the INA3221 power sensor. Power consumption values were logged every 20 milliseconds.

3.4 Real-Time Trajectory Prediction

In this paper, we define real-time trajectory prediction as a model capable of processing and predicting future trajectories at a rate of 30 frames per second (FPS), which corresponds to a latency of 33.3 milliseconds per frame. However, it is important to note that different frame rates can be used depending on the application requirements. We chose 30 FPS as it is a widely accepted standard in this domain and represents the lower bound of what is considered real-time. In many cases, especially for applications requiring higher precision or faster decision-making, frame rates may exceed 30 FPS. Ensuring real-time performance is crucial for applications requiring immediate decision-making, such as autonomous navigation and collision avoidance.

To achieve real-time performance, the batch size must be set to one, ensuring that each frame is processed individually with minimal latency. However, we introduce an alternative definition called real-time throughput, where the model's throughput exceeds 30 frames per second, but a significant delay exists between the input and output. In this

case, while the system processes multiple frames efficiently, the latency per frame may not meet the strict real-time requirement, making it less suitable for applications demanding immediate responsiveness.

We consider a buffer of size B, where incoming data frames are stored until the buffer is full. Once the buffer reaches its capacity, the batch is forwarded to the model for processing. To calculate the maximum latency, let the batch size be B, and assume that frames arrive at time intervals of t_0. The worst-case delay for a frame occurs when it is the first one to enter the buffer, meaning it must wait for the remaining $B-1$ frames to arrive before processing begins. This waiting time is $(B-1)t_0$.

After the batch is complete, it takes an additional T_B time—where T_B is the processing time for a batch of size B—for the output to be generated. Therefore, the total maximum latency for a single frame in this setting is:

$$L_B = (B-1)t_0 + T_B \tag{1}$$

It is important to note that the *real-time throughput* condition ensures that data for a batch is always processed before the buffer overflows. Now, if we cannot meet the real-time frame rate with a batch size of 1, we can check if increasing the batch size to B_1 achieves the real-time throughput. If this condition is met, we can then calculate the latency L_{B_1} for this batch size. Based on this, we can evaluate whether this latency is acceptable for the task at hand. This will be further discussed in the experiments section.

4 Experiments

4.1 Overall Metrics

We evaluated multiple models on different datasets to analyze their performance in terms of prediction accuracy, inference latency, memory usage, and computational efficiency. The models were benchmarked on two different hardware configurations: NVIDIA Quadro RTX 5000 and NVIDIA Jetson Nano Devkit 4GB, both with a batch size of 1. [7, 5, 6]

Model	Dataset	ADE/FDE	Inference Latency	Memory Allocated/Cached	Total Parameters
EqMotion	ETH/UCY	0.40/0.61	57.38 ms ± 1.07 ms	36.45 MB / 38.0 MB	3,027,268
AutoBot(Joint)	Trajnet++	0.128/0.234	10.85 ms ± 0.73 ms	9.21 MB / 14.0 MB	2,409,606
AutoBot(Joint)	nuScenes	- / - [1]	15.59 ms ± 1.25 ms	10.88 MB / 100.0 MB	2,715,270
AutoBot(ego-agent)	nuScenes	1.37/1.03	5.99 ms ± 0.72 ms	5.72 MB / 28.0 MB	1,446,604
MTR	Waymo	0.6697/1.3712	62.43 ms ± 17.70 ms	258.46 MB/1030.0 MB	65,781,334

Table 1: Evaluated Metrics on NVIDIA Quadro RTX 5000 (Batch Size=1)

Model	Dataset	ADE/FDE	Inference Latency	Power Consumption (CPU/GPU/INPUT)
EqMotion	ETH/UCY	0.40/0.61	293.09 ms ± 23.75 ms	732.04 mW / 114.71 mW / 1008.64 mW

Table 2: Evaluated Metrics on NVIDIA Jetson Nano Devkit 4Gb (Batch Size=1)

4.2 Scalability

Increasing batch size has a significant impact on reducing the processing time of trajectory prediction models. As observed in the charts, in all models, larger batch sizes lead to a

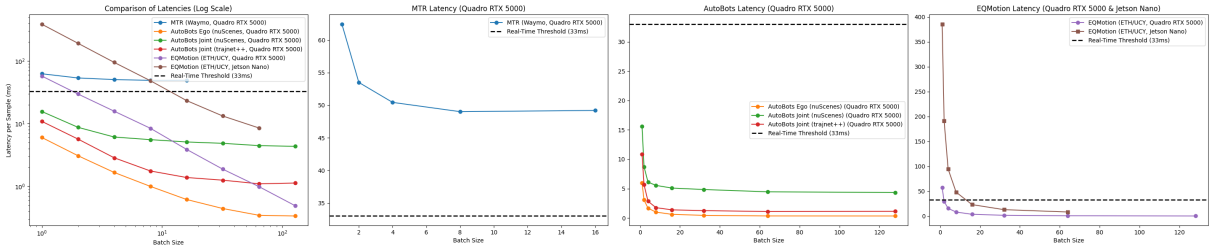

Figure 1: Scalability comparison of trajectory prediction models in terms of latency.

decrease in latency per sample. This reduction occurs due to better utilization of parallel processing in the GPU, which minimizes computational overhead and allows the model to process multiple samples simultaneously. However, the extent of this effect varies depending on the model architecture and dataset complexity.

For instance, in the MTR model, the impact of increasing batch size is less pronounced compared to other models. This is because MTR is a computationally heavy model trained on a large-scale dataset, meaning that even with a small batch size, it already utilizes the GPU's parallel processing capacity to a high degree. As a result, increasing the batch size further does not lead to substantial gains in efficiency. In contrast, models like AutoBots (Ego and Joint) and EqMotion benefit more from larger batch sizes because their computations are relatively lighter, leaving more room for improved parallelization when the batch size increases.

On Jetson Nano, due to its shared memory between CPU and GPU, transferring data between these units requires memory mapping, which increases processing time for small batch sizes. This limitation makes batch size scaling much more effective on this hardware. While increasing batch size is beneficial but limited on more powerful GPUs like Quadro RTX 5000, in Jetson Nano, it becomes a crucial optimization strategy. The improvements in processing time are particularly significant for models like EqMotion, where memory constraints play a key role in performance.

We can also examine real-time throughput for cases that were not real-time in batch size 1. The MTR model did not exhibit real-time throughput in any of the batch sizes we examined, while the Autobot models were real-time. Therefore, we will conduct the analysis on EqMotion. According to Equation 1, we calculate L_B for the batch sizes that have real-time throughput and are lower than the specified threshold (where $t_0 = 33.3$ ms), and we plot the results in Figure 2.

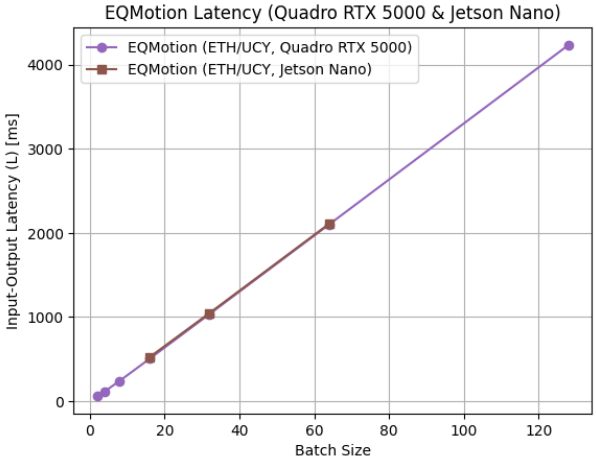

Figure 2: Input-Output latency of EqMotion model for different batch sizes

4.3 Analysis of Energy Consumption

Figure 3 represents the power consumption of the Jetson Nano while running inference using the EqMotion model on the ETH/UCY dataset. Initially, the CPU power consumption increased as the system started processing, followed by the GPU power consumption, which rose when GPU-related operations began. The power usage fluctuates over time, reflecting different stages of the inference process.

On average, during model execution, CPU power consumption was 1668.77 mW, GPU power consumption was 242.20 mW, and input power consumption was 4066.93 mW. In the idle phase after inference, these values dropped to 936.73 mW, 127.49 mW, and 3058.29 mW, respectively. The baseline power consumption when the system was at rest was 732.04 mW (CPU), 114.71 mW (GPU), and 1008.64 mW (input).

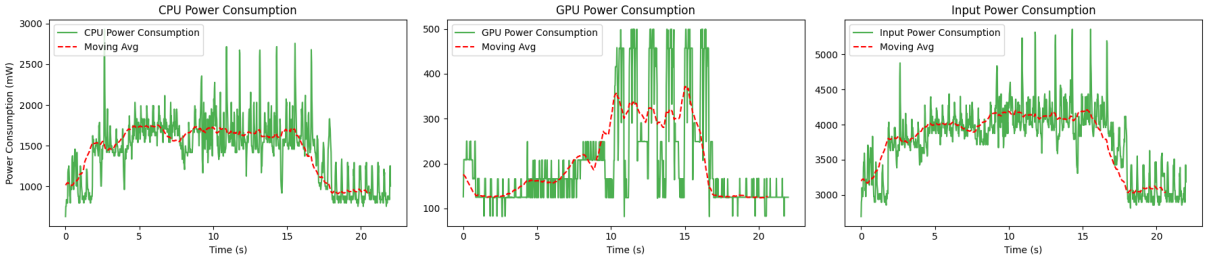

Figure 3: Jetson Nano power consumption during EqMotion inference on the ETH/UCY.

5 Conclusion

In this study, we evaluated multiple trajectory prediction models across different datasets and hardware platforms, analyzing their trade-offs between computational efficiency and accuracy. Our results highlight that while high-performance GPUs such as the NVIDIA Quadro RTX 5000 enable fast inference with minimal latency, edge computing devices like the NVIDIA Jetson Nano present significant constraints in memory usage and inference speed.

From our findings, models like AutoBot (Joint) achieve a strong balance between low ADE/FDE and moderate computational costs, making them suitable for real-time

applications. However, complex models such as MTR demonstrate superior accuracy at the cost of significantly higher computational demands, limiting their feasibility for resource-constrained environments.

These results emphasize the importance of selecting appropriate models based on deployment requirements. Future work can explore optimization techniques, such as model quantization and pruning, to enhance the efficiency of high-performing models for edge computing scenarios. Additionally, extending this evaluation to more diverse datasets, baseline models and real-world applications will further validate the practical implications of trajectory prediction models.

6 References

References

[1] M. Nicolella, S. Roozkhosh, D. Hoornaert, A. Bastoni, and R. Mancuso, "Rt-bench: an extensible benchmark framework for the analysis and management of real-time applications," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, ser. RTNS 2022. ACM, Jun. 2022, p. 184–195. [Online]. Available: http://dx.doi.org/10.1145/3534879.3534888

[2] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," 2020. [Online]. Available: https://arxiv.org/abs/1911.02549

[3] Z. Cheng, J. Kasai, and T. Yu, "Batch prompting: Efficient inference with large language model apis," 2023. [Online]. Available: https://arxiv.org/abs/2301.08721

[4] L. Feng, M. Bahari, K. M. B. Amor, É. Zablocki, M. Cord, and A. Alahi, "Unitraj: A unified framework for scalable vehicle trajectory prediction," *arXiv preprint arXiv:2403.15098*, 2024.

[5] R. Girgis, F. Golemo, F. Codevilla, M. Weiss, J. A. D'Souza, S. E. Kahou, F. Heide, and C. Pal, "Latent variable sequential set transformers for joint multi-agent motion prediction," 2022. [Online]. Available: https://arxiv.org/abs/2104.00563

[6] S. Shi, L. Jiang, D. Dai, and B. Schiele, "Motion transformer with global intention localization and local movement refinement," 2023. [Online]. Available: https://arxiv.org/abs/2209.13508

[7] C. Xu, R. T. Tan, Y. Tan, S. Chen, Y. G. Wang, X. Wang, and Y. Wang, "Eqmotion: Equivariant multi-agent motion prediction with invariant interaction reasoning," 2023. [Online]. Available: https://arxiv.org/abs/2303.10876