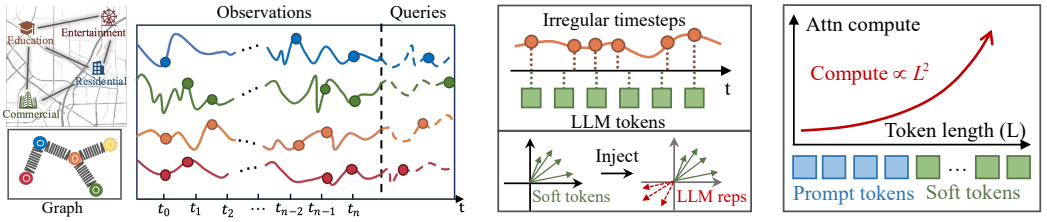
LLMODE: ALIGNING ODES WITH LLMS VIA GATED TOKEN INJECTION FOR IRREGULAR SPATIO-TEMPORAL FORECASTING

Anonymous authors
Paper under double-blind review

Large language models (LLMs) have shown promise for spatio-temporal forecasting, but existing approaches often rely on regularly sampled token sequences and struggle with irregular observations because of temporal asynchrony, representation-space misalignment, and limited context windows. We propose LLMODE, a token-efficient framework for irregular spatio-temporal forecasting with a frozen LLM backbone. LLMODE first uses a graph-aware ODE encoder to reconstruct irregular graph observations as a continuous-time latent trajectory. A Fixed-Budget Perceiver Resampler then compresses this variable-length trajectory into a fixed number of dynamic memory tokens. In parallel, compact statistical descriptors are encoded and resampled into context memory tokens. A dual-source gated cross-attention module injects both memories into the frozen LLM, enabling controlled utilization of external spatio-temporal evidence. Experiments on three real-world urban datasets and two physical-dynamics benchmarks show competitive overall performance, with clearer advantages under sparse or dynamically complex irregular sampling. Additional evaluations on unseen urban regions further demonstrate strong zero-shot generalization without adaptation. Our code and sample data are attached for anonymous review: `https://anonymous.4open.science/r/LLM4TS-55FA`.

1 INTRODUCTION

Irregularly sampled spatio-temporal data are common in urban sensing and physical systems, as illustrated in Figure 1. Traffic flows, crime reports, mobility records, and interacting entities may be observed at non-uniform timestamps, with missing values and asynchronous updates across locations or objects. Because these observations evolve over relational structures, forecasting must jointly model temporal irregularity, incomplete measurements, and graph-structured dependencies.



(a) Problem setup (b) Semantic alignment (c) Context-length constraint

Figure 1: (a) Problem setup for irregular spatio-temporal graph forecasting. (b) Challenge 1: Semantic alignment between soft tokens and LLM semantic space. "LLM reps" refer to the representations in the LLM's hidden space. (c) Challenge 2: Context-length constraint in conditioning LLMs.

Large language models (LLMs) have recently been applied to spatio-temporal forecasting through token-based conditioning. UrbanGPT Li et al. (2024a), for example, compresses spatio-temporal signals into soft tokens for a pretrained LLM. However, most LLM-based forecasters assume regularly sampled sequences and fixed-step tokenization, which obscure actual time gaps, missing intervals, and asynchronous node updates. ISTS-PLM Zhang et al. (2025) introduces interval-aware time encodings for irregular time series, but remains token-discrete and primarily targets individual sequences. Extending such methods to graph-based forecasting is therefore non-trivial because spatial coupling and temporal irregularity must be handled simultaneously.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Neural ODEs offer a natural alternative by evolving latent states according to actual elapsed time, reconstructing trajectories between sparse observations, and supporting queries at arbitrary timestamps. Their integration with frozen LLMs nevertheless raises two challenges. First, ODE encoders produce long trajectories containing many locally smooth and redundant neighboring states. Directly exposing all states increases the token budget and attention cost while potentially diluting informative transitions. Second, compressed numerical tokens remain misaligned with the representation space learned through language pretraining. Direct concatenation neither explicitly retrieves evidence relevant to a prediction request nor controls how strongly that evidence modifies frozen hidden representations. Effective integration therefore requires compact memory construction and controlled, prediction-aware evidence injection.

To address these challenges, we propose LLMODE, an ODE-enhanced frozen-LLM framework for irregular spatio-temporal forecasting. A graph-aware ODE encoder reconstructs asynchronous and partially observed graph sequences as a continuous-time latent trajectory without imposing a regular temporal grid. A Fixed-Budget Perceiver Resampler compresses this trajectory into dynamic memory tokens, using a dynamics-aware attention bias and explicit time labels to preserve informative changes and their temporal locations. In parallel, compact statistical descriptors are encoded and resampled into context memory tokens that provide complementary global information. A dual-source gated cross-attention module then injects both memories into selected layers of the frozen LLM. For each requested prediction timestamp, the LLM prompt contains a dedicated query-slot token, whose final hidden state is mapped to the corresponding forecast. Experiments against 11 baselines on three urban datasets and two physical-dynamics benchmarks show competitive overall performance, with more pronounced advantages under sparse or dynamically complex observations. On NYCbike zero-shot, LLMODE reduces token usage by 89.2% and achieves a 7.7$\times$ inference speedup over UrbanGPT while maintaining strong generalization to unseen regions.

Our main contributions are as follows.

- We formulate irregular spatio-temporal graph forecasting with a frozen LLM and identify temporal irregularity, representation-space alignment, and context-length constraints as the central challenges.
- We propose **LLMODE**, which combines a graph-aware ODE encoder with a Fixed-Budget Perceiver Resampler to reconstruct continuous-time graph dynamics and compress them into dynamic memory tokens under a bounded token budget.
- We introduce dual-source gated cross-attention with gated feed-forward refinement to integrate dynamic and context memories into a frozen LLM in a controlled manner.
- Extensive experiments on urban and physical-dynamics benchmarks demonstrate competitive forecasting performance, improved robustness under sparse irregular observations, and strong zero-shot generalization to unseen regions.

2 Problem Formulation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a fixed graph, where $\mathcal{V}$ and $\mathcal{E}$ represent the node and edge sets, respectively, and $|\mathcal{V}| = N$ is the number of nodes. Each node $v \in \mathcal{V}$ is associated with a D-dimensional observation vector $\mathbf{x}_v(t) \in \mathbb{R}^D$ at continuous time t, where D denotes the number of features.

Instead of observing all nodes on a shared regular time grid, we collect an irregular observation set:

$$\mathcal{D}_{\text{obs}} = \{(v, t_{v,i}, \mathbf{x}_v(t_{v,i}))\}, \tag{1}$$

where $t_{v,i}$ denotes the i-th observation timestamp of node v and $\mathbf{x}_v(t_{v,i})$ is the corresponding observed feature vector. The observation intervals $t_{v,i+1} - t_{v,i}$ are non-uniform, and different nodes may have different timestamp sets.

Given a set of query timestamps $\mathcal{Q}_v = \{t_{v,n}^{(q)}\}_{n=1}^{L_q}$ for node v, where L_q denotes the number of query points, the goal is to jointly predict a multivariate target vector at each arbitrary query time:

$$\widehat{\mathbf{y}}_v\left(t_{v,n}^{(q)}\right) = f\left(\mathcal{G}, \mathcal{D}_{\text{obs}}, v, t_{v,n}^{(q)}\right) \in \mathbb{R}^D, \tag{2}$$

where D denotes the number of jointly predicted target variables, and $\widehat{\mathbf{y}}_v(t_{v,n}^{(q)})$ is the predicted target vector of node v at query time $t_{v,n}^{(q)}$.

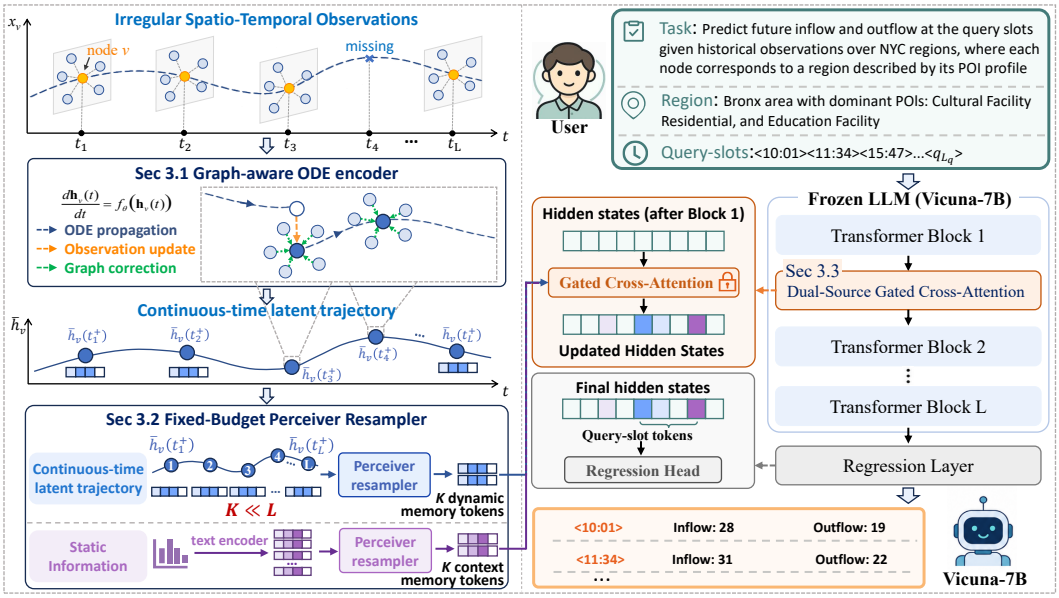


Figure 2: **Framework of LLMODE.** LLMODE constructs two memory streams to condition a frozen LLM backbone. The graph-aware ODE encoder first reconstructs a continuous-time latent trajectory, which is resampled into a fixed number of dynamic memory tokens. In parallel, compact statistical descriptors are encoded and resampled into context memory tokens. Prompt hidden states retrieve information from both memories through dual-source gated cross-attention, and the final query-slot states are mapped to forecasts.

3 METHOD

As illustrated in Figure 2, LLMODE comprises a graph-aware ODE encoder that reconstructs continuous-time graph dynamics, a Fixed-Budget Perceiver Resampler that compresses dynamic trajectories and context descriptors into fixed-size memories, and a dual-source gated cross-attention module that injects both memories into the frozen LLM for forecasting.

3.1 GRAPH-AWARE ODE ENCODER

Irregular sampling and missing observations create nonuniform time intervals during which the underlying system continues to evolve without direct measurements. We therefore represent each node v by a continuous latent state governed by

$$\frac{d\mathbf{h}_v(t)}{dt} = f_\theta(\mathbf{h}_v(t)), \tag{3}$$

where $\mathbf{h}_v(t) \in \mathbb{R}^{d_{\text{ode}}}$ denotes the latent state of node v, and f_θ is a learnable vector field instantiated using a GRU-ODE parameterization. An ODE solver propagates the latent state over the actual elapsed interval between observations.

When an observation $\mathbf{x}_{v,i} \in \mathbb{R}^D$ arrives at time t_i, the ODE solver first produces the propagated state $\mathbf{h}_v(t_i^-)$. Because continuous propagation may gradually deviate from the observed trajectory, we then use the newly arrived observation to correct this state:

$$\mathbf{h}_v(t_i^+) = \text{GRU}\big(\mathbf{h}_v(t_i^-), \Phi\big(\mathbf{x}_{v,i}, \mathbf{h}_v(t_i^-)\big)\big), \tag{4}$$

where the superscripts $-$ and $+$ denote the latent states immediately before and after observation correction, respectively. The observation encoder Φ constructs the correction signal from the new observation and its discrepancy from the current latent prediction.

To capture structural interactions, the corrected node states $\{\mathbf{h}_v(t_i^+)\}_{v\in\mathcal{V}}$ are further refined through a gated graph-based residual update. The resulting graph-aware state $\bar{\mathbf{h}}_v(t_i^+)$ is used as the initial

Table 1: Main results on NYC and physics benchmarks, measured by MAE and MSE. Results are averaged over 5 independent runs. Overall, LLMODE achieves an average relative improvement of 6.3% across all datasets and evaluation metrics. $*$ indicates the statistically significant improvements over the best baseline, with p-value smaller than 0.001.

Method	NYCtaxi				NYCbike				NYCcrime				Springs				Charged			
	inflow		outflow		inflow		outflow		robbery		burglary		X		Y		X		Y	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
TGCN	0.1145	0.0367	0.0875	0.0331	0.2211	0.1572	0.1506	0.0657	0.9970	1.0849	0.6746	1.0029	0.1971	0.0741	0.1934	0.0708	0.4706	0.3038	0.4710	0.3042
STSGCN	0.0847	0.0286	0.0704	0.0349	0.1951	0.1734	0.1669	0.1119	0.9162	1.5371	0.6293	0.8300	0.3306	0.1686	0.3114	0.1754	0.3804	0.2153	0.4152	0.2879
MTGNN	0.0371	0.0143	0.0337	0.0365	0.0786	0.0263	0.0824	0.0265	0.8252	1.1408	0.6283	0.7857	0.1053	0.0237	0.1086	0.0249	0.1410	0.0533	0.1482	0.0535
BiTGraph	0.0731	0.0305	0.0540	0.0301	0.0837	0.0298	0.0830	0.0275	0.8555	1.2376	0.5321	0.4988	0.1271	0.0312	0.1290	0.0321	0.1501	0.0578	0.1527	0.0593
FourierGNN	0.1234	0.0246	0.1026	0.0187	0.1533	0.0734	0.1523	0.0626	0.4074	0.8993	0.2666	0.2899	0.2108	0.0767	0.2127	0.0788	0.2305	0.1000	0.2309	0.1003
ISTS-PLM	**0.0329**	0.0068	0.0263	0.0059	0.1061	0.0461	0.1077	0.0488	0.4271	0.4055	0.3040	0.2143	0.1513	0.0427	0.1505	0.0423	0.1524	0.0569	0.1539	0.0576
UrbanGPT	0.0359	0.0080	0.0262	0.0068	0.0787	0.0301	0.0833	0.0292	0.9336	1.7088	0.6192	0.8341	0.1519	0.0420	0.1514	0.0424	0.1633	0.0702	0.1746	0.0908
GPT4TS	0.0424	0.0260	0.0312	0.0209	0.1592	0.2072	0.1636	0.2129	0.3350	0.2578	0.2398	0.1229	0.4549	0.2826	0.4549	0.2838	0.4996	0.3447	0.5009	0.3469
GRU-ODE	0.0956	0.0645	0.0708	0.0607	0.3100	0.4081	0.3443	0.4964	0.3678	0.4161	0.2483	0.1815	0.3132	0.1664	0.3080	0.1615	0.3001	0.1543	0.2888	0.1399
tPatchGNN	0.0338	0.0082	0.0347	0.0095	0.0800	0.0476	0.0849	0.0484	0.3052	0.1428	0.2483	0.1119	0.1391	0.0370	0.1384	0.0367	0.1439	0.0504	0.1497	0.0504
ViTST	0.3548	0.1718	0.2938	0.1377	0.4371	0.6891	0.4542	0.7136	0.4582	0.5084	0.3267	0.2325	0.4519	0.2751	0.4524	0.2767	0.4842	0.3174	0.4849	0.3181
LLMODE	0.0336	**0.0066**	**0.0252**	**0.0057**	**0.0767***	**0.0243***	**0.0817**	**0.0242***	**0.3050**	**0.1396***	**0.2394**	**0.0790***	**0.0973***	**0.0191***	**0.0986***	**0.0196***	**0.1402**	**0.0483***	**0.1461***	**0.0493***

state for continuous propagation over the next interval. The detailed GRU-ODE parameterization, observation encoding, and graph-aware correction are provided in Appendix A.4–A.6.

At a set of solver timestamps $\{\tau_l\}_{l=1}^{L}$, the states of all nodes are collected as $\bar{\mathbf{H}}(\tau_l) \in \mathbb{R}^{N\times d_{\text{ode}}}$ and stacked along the temporal dimension to construct the continuous-time latent trajectory

$$\mathbf{H}_{\text{dyn}} = \left[\bar{\mathbf{H}}(\tau_1); \ldots; \bar{\mathbf{H}}(\tau_L)\right] \in \mathbb{R}^{N\times L\times d_{\text{ode}}}. \tag{5}$$

The resulting trajectory captures the continuous temporal evolution and structural interactions of all nodes and serves as the input to the subsequent fixed-budget resampler.

3.2 Fixed-Budget Perceiver Resampler

Given an irregularly sampled input trajectory, the graph-aware ODE encoder uses its smallest observation gap as the ODE integration step and records the propagated latent states as $\mathbf{H}_{\text{dyn}} \in \mathbb{R}^{N\times L\times d_{\text{ode}}}$. These states are projected into the resampler space $\mathbb{R}^{d_{\text{mem}}}$ before resampling. Although the reconstructed trajectory provides fine-grained dynamic states, directly injecting all L states is computationally expensive and may dilute informative changes because adjacent states can be highly redundant.

We therefore use a Perceiver Resampler to distill each reconstructed latent trajectory into a fixed number of dynamic memory tokens. For clarity, we omit the node index v in the following derivation and describe the resampling for a single node. Specifically, the resampler uses K learnable latent queries to attend to the input sequence and outputs $\{\mathbf{z}_k\}_{k=1}^{K}$, where $\mathbf{z}_k \in \mathbb{R}^{d_{\text{mem}}}$. Rapidly changing intervals generally contain more informative dynamic evidence. However, standard attention alone does not explicitly encourage the resampler to preserve states from these intervals. We therefore introduce the magnitude of the latent dynamics directly into the attention logits:

$$\alpha_{k,i} = \text{softmax}_i\left(\frac{\mathbf{q}_k^\top \mathbf{k}_i}{\sqrt{d_{\text{mem}}}} + \beta\ \text{Norm}\Big(\Big\|\dot{\mathbf{h}}(t_i)\Big\|_2\Big)\right). \tag{6}$$

Here, $\alpha_{k,i}$ is the attention weight from the k-th latent query to the state at timestamp t_i; $\mathbf{q}_k \in \mathbb{R}^{d_{\text{mem}}}$ is a learnable latent query, and $\mathbf{k}_i \in \mathbb{R}^{d_{\text{mem}}}$ is the key representation obtained from the latent state $\mathbf{h}(t_i)$ through a learnable projection; d_{mem} denotes the resampler and memory-token dimension; $\dot{\mathbf{h}}(t_i) = f_\theta(\mathbf{h}(t_i))$ represents the instantaneous latent change given by the ODE vector field; $\text{Norm}(\cdot)$ denotes normalization over the latent trajectory; and β controls the strength of the dynamics-aware bias. This bias encourages the resampler to retain states from intervals with more pronounced changes while preserving content-based attention.

The downstream forecasting prompt specifies the timestamps to be predicted. To align the distilled dynamic memory content with these query times, we assign each resampled vector an explicit time label derived from the same attention weights. The time label is fused with $\mathbf{z}_k$ to form the time-aware representation $\tilde{\mathbf{z}}_k$. Collecting the K tokens yields the dynamic memory

$$\tilde{\mathbf{Z}} = [\tilde{\mathbf{z}}_1; \ldots; \tilde{\mathbf{z}}_K] \in \mathbb{R}^{K\times d_{\text{mem}}}. \tag{7}$$

The detailed resampling and time-labeling process is provided in Appendix A.7– A.8.

3.3 DUAL-SOURCE GATED CROSS-ATTENTION

In addition to the dynamic memory $\widetilde{\mathbf{Z}}$ extracted from the continuous-time latent trajectory, we construct a context memory $\mathbf{P}$ from the observed history. The observations are summarized as compact statistical descriptions, encoded by a language encoder, and compressed by a resampler into K context memory tokens, $\mathbf{P} \in \mathbb{R}^{K \times d_{\text{mem}}}$. While $\widetilde{\mathbf{Z}}$ captures fine-grained temporal dynamics, $\mathbf{P}$ provides complementary global statistics. Details of the context-memory construction are provided in Appendix A.9.

The two memories are injected into the frozen LLM through dual-source gated cross-attention, avoiding the uncontrolled mixing caused by direct prompt concatenation. A forecasting prompt specifies the task and arbitrary query timestamps. Representative prompts are provided in Appendix A.10. After a Transformer layer, its hidden states $\mathbf{Y}$ are updated as follows:

$$\mathbf{\Delta}_{\text{mem}} = \lambda \,\text{CrossAttn}\left(\mathbf{Y}, \widetilde{\mathbf{Z}}\right) + (1-\lambda)\,\text{CrossAttn}\left(\mathbf{Y}, \mathbf{P}\right). \tag{8}$$

$$\widehat{\mathbf{Y}} = \mathbf{Y} + \alpha_{\text{attn}} \mathbf{\Delta}_{\text{mem}}. \tag{9}$$

$$\mathbf{Y}' = \widehat{\mathbf{Y}} + \alpha_{\text{ffn}}\,\text{FFN}\left(\widehat{\mathbf{Y}}\right), \tag{10}$$

where

$$\begin{aligned} \alpha_{\text{attn}} &= \tanh\left(g_{\text{attn}}\right), \\ \alpha_{\text{ffn}} &= \tanh\left(g_{\text{ffn}}\right). \end{aligned} \tag{11}$$

Here, $\mathbf{Y}, \widehat{\mathbf{Y}}, \mathbf{Y}' \in \mathbb{R}^{L_p \times d_{\text{LLM}}}$ denote the prompt-token hidden states before injection, after cross-attention, and after feed-forward refinement, respectively. Among the L_p prompt tokens, L_q serve as query slots. $\text{CrossAttn}(\mathbf{Y}, \mathbf{M})$ projects the prompt states and memory tokens into a shared attention space, using them as queries and keys/values, respectively, and returns an update in $\mathbb{R}^{L_p \times d_{\text{LLM}}}$, while λ balances the two memories. The learnable parameters g_{attn} and g_{ffn} are initialized to zero, making the module an identity mapping at the beginning of training.

For Vicuna-7B, we insert two such modules after Transformer Layers 1 and 16. The updated states are propagated through the remaining layers. Each query-slot token corresponds to a prediction timestamp, and its final-layer hidden state is extracted and projected by a lightweight regression head to produce the corresponding forecast. The model is trained using MSE, with the complete training configuration provided in Appendix A.2.

4 EXPERIMENTS

Our experiments are designed to answer the following questions:

Q1: Overall effectiveness. Can LLMODE achieve competitive forecasting performance across both real-world urban datasets and physical-dynamics benchmarks?

Q2: Unseen-region generalization. Can LLMODE transfer to spatial regions that are entirely unseen during training and forecast their future states without additional adaptation?

Q3: Necessity of continuous-time modeling. Does continuous-time encoding become increasingly important as observations become sparser, compared with discrete spatio-temporal encoders and existing baselines?

Q4: Effectiveness of fixed-budget dynamic evidence. Can a small number of adaptively selected dynamic states preserve the forecasting-relevant information in the continuous trajectory more effectively than exposing the full trajectory or applying simple compression?

Q5: Contribution and utilization of the frozen LLM. First, do pretrained frozen representations provide benefits beyond a prediction network or a Transformer trained from scratch? Second, does the frozen backbone effectively use the injected dynamic and contextual evidence, and does controlled gated injection improve this utilization?

We first describe the experimental setup and then address Q1–Q5, followed by a qualitative case study.

4.1 Experimental Setup

Datasets and protocols. We use NYCtaxi, NYCbike, and NYCcrime for urban forecasting, and Springs and Charged for particle dynamics. NYCtaxi/NYCbike use 24-hour histories to predict 12 hours; NYCcrime uses 96 days to predict 48 days. The supervised setting evaluates training regions, while the zero-shot setting holds out 80 NYC regions. Springs and Charged predict future 2D particle positions from asynchronous partial observations. Dataset construction and preprocessing details are provided in Appendix A.1.

Baselines. We compare against classical spatio-temporal models, including TGCN, STSGCN, MTGNN, BiTGraph, and FourierGNN; LLM-based forecasters, including ISTS-PLM, UrbanGPT, and GPT4TS; and irregular-aware methods, including GRU-ODE, tPatchGNN, and ViTST Zhao et al. (2019); Song et al. (2020); Wu et al. (2020); Chen et al. (2023); Yi et al. (2023); Zhang et al. (2025); Li et al. (2024a); Zhou et al. (2023); De Brouwer et al. (2019); Zhang et al. (2024); Li et al. (2023). Regular-grid baselines use linearly interpolated inputs. Detailed descriptions of the baselines are provided in Appendix A.16.

Implementation. We use frozen Vicuna-7B Zheng et al. (2023) as the default backbone. Model configurations, optimization settings, and hardware information are provided in Appendix A.3.

4.2 Q1: Overall Forecasting Effectiveness

To answer Q1, Table 1 compares LLMODE with conventional spatio-temporal, irregular-aware, and LLM-based forecasters on three real-world urban datasets and two physical-dynamics benchmarks.

Overall, LLMODE achieves competitive or superior performance across all five benchmarks. The strongest gains are observed on NYCcrime, Springs, and Charged, where observations are more event-driven, sparse, or governed by complex interactions. This result answers Q1 affirmatively and indicates that the complete framework is not restricted to either urban data or physical systems. The smaller gains on the denser mobility datasets are further analyzed in Appendix A.11.

4.3 Q2: Unseen-Region Generalization

To answer Q2, we evaluate LLMODE under the zero-shot protocol on the NYC benchmarks. The model is trained on 80 regions and directly tested on another 80 unseen regions without fine-tuning. This setting tests whether LLMODE can infer future dynamics for unseen regions from their observed histories, rather than relying on region-specific patterns learned during training.

As shown in Table 2, LLMODE achieves the best performance on five of the six metrics. The gains are particularly clear on the much sparser NYCcrime dataset, indicating effective generalization to unseen regions under sparse and irregular observations. The contribution of the frozen LLM to this transferability is further analyzed in Q5.

Table 2: Zero-shot performance on NYC benchmarks, averaged over target variables and 5 runs. Arrows indicate relative changes from the strongest baseline. Lower is better.

Method	NYCtaxi		NYCbike		NYCcrime	
	MAE	MSE	MAE	MSE	MAE	MSE
TGCN	0.3233	0.6351	0.2457	0.2105	0.7266	2.6862
STSGCN	0.2758	0.4385	0.2221	0.2723	0.7431	2.8640
MTGNN	0.0878	0.0748	0.0769	0.0357	0.5574	1.7864
BiTGraph	0.0964	<u>0.0563</u>	0.0880	0.0415	0.4703	1.5098
FourierGNN	0.1538	0.0628	0.1528	0.0680	0.5304	1.4962
ISTS-PLM	0.1042	0.0931	0.0822	0.0371	0.2041	<u>0.3421</u>
UrbanGPT	**0.0798**	0.0634	0.0778	<u>0.0292</u>	0.7418	2.0739
GPT4TS	0.1210	0.1524	0.1017	0.0834	0.2885	0.8108
GRU-ODE	0.2510	0.4434	0.2348	0.2410	0.4219	1.6564
tPatchGNN	0.0830	0.0577	<u>0.0765</u>	0.0323	<u>0.1644</u>	0.3863
ViTST	0.4427	0.5576	0.3676	0.3013	0.8560	2.7815
LLMODE (Ours)	<u>0.0801</u> ↑ 0.4%	**0.0396** ↓ 29.7%	**0.0745** ↓ 2.6%	**0.0260** ↓ 11.0%	**0.1527** ↓ 7.1%	**0.2791** ↓ 18.4%

4.4 Q3: Necessity of Continuous-Time Modeling

To answer Q3, Figure 3 compares LLMODE with representative methods under controlled missing rates. ST-Encoder replaces the graph-aware ODE encoder with the discrete spatio-temporal encoder used in UrbanGPT, while keeping all other components unchanged. The remaining methods include representative regular- and irregular-sampling baselines.

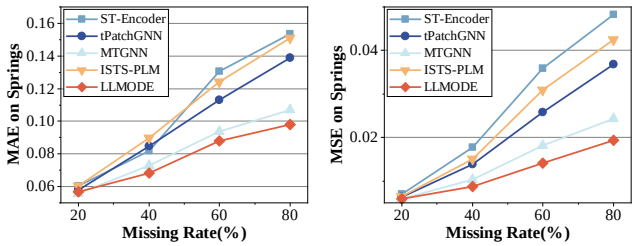


Figure 3: Forecasting performance under different missing rates on the Springs system.

As the missing rate increases, ST-Encoder degrades more rapidly than LLMODE, indicating the limitation of discrete encoders under sparse observations. MTGNN remains relatively robust due to its adaptive graph and multi-scale temporal modeling, whereas tPatchGNN and ISTS-PLM also show increasing errors. In contrast, LLMODE consistently achieves the lowest errors, with a widening advantage at higher missing rates. These results support the importance of continuous-time modeling for reconstructing dynamics over long unobserved intervals. Solver and step-size sensitivity analyses are provided in Appendix A.12.

4.5 Q4: Effectiveness of Fixed-Budget Dynamic Evidence

To answer Q4, we examine both the token budget and the trajectory-state selection strategy. The continuous-time encoder reconstructs each trajectory into $L = 120$ latent states, and Figure 4 compares different fixed-budget settings and full-trajectory exposure, with UrbanGPT included as an LLM-based reference.

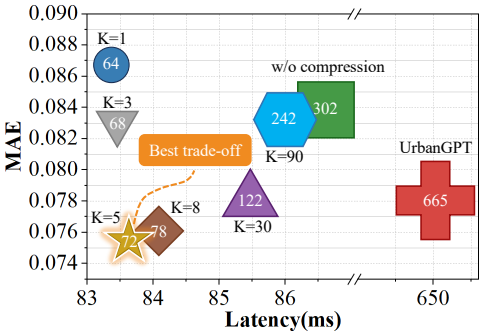


Figure 4: Token-budget efficiency on NYCbike zero-shot. Numbers inside markers indicate token counts.

Compared with UrbanGPT, LLMODE reduces the reported token budget by nearly 90% and inference latency by approximately 87%, while achieving better zero-shot accuracy. An extremely small budget ($K = 1$) loses substantial trajectory information, whereas $K = 5$ and $K = 8$ achieve the strongest accuracy–efficiency trade-off. Across datasets and task settings, these two budgets consistently perform best, indicating that a small fixed token budget is robust across different forecasting settings. Larger budgets and full-trajectory exposure yield no consistent gains, suggesting that forecasting-relevant information is concentrated in a small subset of states. Detailed results, together with the inference latency of representative non-LLM methods for reference, are provided in Appendix A.13. Figure 5 further compares learned resampling with random selection and linear projection. Learned resampling consistently performs better, indicating that adaptive state selection is more effective than simple sequence reduction. Removing either the dynamics-aware bias or explicit time labels also degrades performance, confirming that compact dynamic memory must preserve both salient state changes and their temporal locations.

4.6 Q5: Frozen-LLM Contribution and Evidence Utilization

To answer Q5, we compare the frozen LLM with an MLP and a randomly initialized six-layer Transformer to examine the benefits of pretrained representations beyond trainable models. We further evaluate different frozen backbones, evidence injection mechanisms, and evidence contents to assess the robustness of the backbone and the effectiveness of evidence utilization.

Contribution of the Frozen LLM. As shown in Table 3, replacing the frozen LLM with trainable alternatives, including an MLP prediction head and a randomly initialized Transformer, achieves comparable supervised performance but degrades zero-shot performance. This suggests that train-

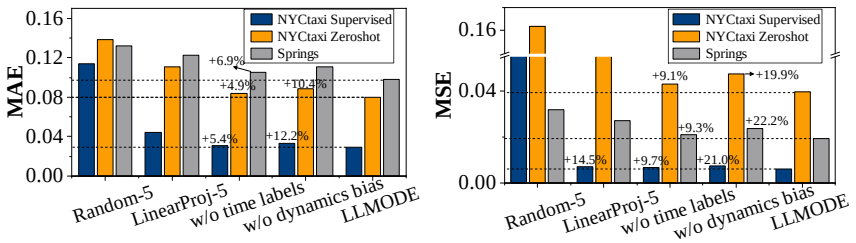


Figure 5: Comparison of different evidence-selection mechanisms under a fixed token budget. The left and right panels report MAE and MSE, respectively.

able models can capture forecasting patterns from observed regions, whereas the pretrained LLM provides additional modeling ability that is particularly important for transfer to unseen regions.

Moreover, the three frozen LLM backbones achieve comparable performance on both seen and unseen regions. This consistency shows that the benefit of the frozen LLM is not specific to Vicuna-7B, and that LLMODE can effectively work with different pretrained backbones.

Table 3: Contribution and robustness analysis of the frozen LLM. Percentage changes are relative to the default Vicuna-7B backbone.

Model	NYCtaxi Supervised		NYCtaxi Zero-shot	
	MAE	MSE	MAE	MSE
Pretraining and capacity controls				
Ours w/ MLP	0.0317↑ 7.2%	0.0071↑ 12.6%	0.1131↑ 41.2%	0.1048↑ 164.6%
Ours w/ random Transformer	0.0301↑ 2.4%	0.0065↑ 4.8%	0.0925↑ 15.5%	0.0521↑ 31.6%
Frozen-backbone robustness				
Best baseline	0.0296↑ 0.7%	0.0064↑ 3.2%	0.0798↓ 0.4%	0.0563↑ 42.2%
Ours w/ Llama-3.1-8B	**0.0288**↓ 2.0%	**0.0060**↓ 3.2%	0.0804↑ 0.4%	0.0378↓ 4.5%
Ours w/ Llama-2-13B	0.0296↑ 0.7%	0.0067↑ 8.1%	**0.0793**↓ 1.0%	**0.0371**↓ 6.3%
Ours w/ Vicuna-7B (default)	0.0294	0.0062	0.0801	0.0396

Evidence Utilization. As shown in Figure 6, cross-attention outperforms prompt concatenation, and learnable gating provides further gains, confirming the benefit of controlled retrieval from separate memories. Figure 7 shows that the gate gains α_{attn} and α_{ffn}, together with their update contributions, increase from zero and stabilize during training. Details of the Effective Contribution metric and additional insertion-frequency experiments are provided in Appendix A.14.

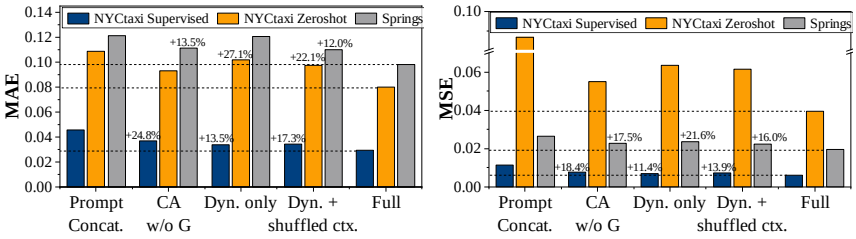


Figure 6: Ablation study of evidence access mechanisms and evidence sources. "Prompt Concat." denotes prompt concatenation, "CA w/o G" denotes cross-attention without gating, "Dyn. only" uses only dynamic memory, and "Dyn. + shuffled ctx." uses dynamic memory with randomly shuffled context memory.

The source ablations show that dynamic memory alone is insufficient and that shuffled context degrades performance, showing that sample-specific context provides complementary information rather than merely additional tokens. Overall, the frozen LLM effectively utilizes both memory sources, with gated injection further improving their integration.

4.7 Qualitative Case Study

Figure 8 compares LLMODE with tPatchGNN (Zhang et al., 2024), a strong baseline for irregular temporal modeling, on a representative supervised NYCtaxi case. LLMODE follows the ground-truth trajectory more closely and better captures the magnitude and timing of peak variations. This behavior is consistent with its continuous-time modeling, fixed-budget dynamic memory, and gated injection design. For visualization, predictions from multiple samples are concatenated, with details provided in Appendix A.15.

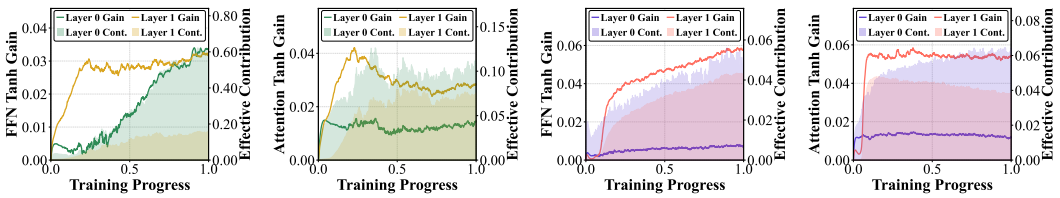
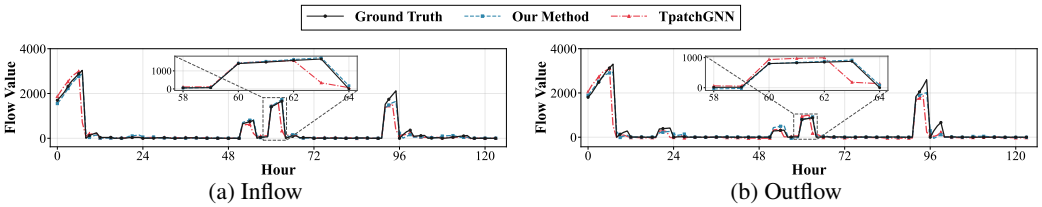


Figure 7: Training-time evolution of the global injection gates on the NYC datasets. Layer 0 and Layer 1 denote the two injection modules; "Cont." denotes the normalized contribution proxy.



(a) Inflow (b) Outflow

Figure 8: Qualitative comparison on NYCtaxi over a 124-hour evaluation period. The enlarged region highlights a representative demand peak. LLMODE more closely follows the ground-truth temporal patterns than tPatchGNN, particularly around rapid variations.

5 Related Work

Spatio-temporal Forecasting. Spatio-temporal forecasting models temporal and spatial dependencies using recurrent, convolutional, and graph-based architectures Yu et al. (2017); Zhang et al. (2017); Zhao et al. (2019); Song et al. (2020); Wu et al. (2020; 2019). Recent work further explores adaptive graphs and multi-scale or frequency-aware representations Jin et al. (2023b); Ye et al. (2022); Yuan et al. (2024); Dong et al. (2024); Wang et al. (2024a); Chen et al. (2023); Yi et al. (2023). However, most methods assume regular sampling and supervised settings, leaving severe irregularity and zero-shot transfer underexplored.

LLM-based Forecasting. Recent LLM-based forecasting methods use soft-token injection, textualization, prompting, projection, adapters, or partial fine-tuning to align temporal signals with LLM representations Li et al. (2024a); Xue & Salim (2023); Cao et al. (2024); Wang et al. (2024b); Pan et al. (2024); Hu et al. (2025); Chang et al. (2025); Sun et al. (2024); Jin et al. (2023a); Liu et al. (2025); Jiang et al. (2025); Liu et al. (2024). Cross-attention has also been used for external-feature injection in multimodal models such as Flamingo Alayrac et al. (2022). However, irregular and asynchronous spatio-temporal evidence with explicit spatial dependencies remains underexplored.

Irregularly Sampled Time Series Modeling. Irregular time series are handled by interpolation or resampling Lipton et al. (2016), time-aware recurrent networks Neil et al. (2016); Che et al. (2018); Baytas et al. (2017); Jhin et al. (2024), continuous-time neural ODE models Chen et al. (2018); Rubanova et al. (2019); De Brouwer et al. (2019); Huang et al. (2020); Biloš et al. (2021); Kidger et al. (2020); Dupont et al. (2019), and attention-based methods Zhang et al. (2024); Li et al. (2023). Interpolation may distort dynamics under sparse observations Shukla & Marlin (2021), while LLM-based irregular spatio-temporal forecasting remains underexplored.

6 Conclusion

In this work, we presented LLMODE, a frozen-LLM framework for irregular spatio-temporal forecasting. LLMODE reconstructs continuous-time graph dynamics from asynchronous and partially observed data, distills long latent trajectories into fixed-budget dynamic memory, and injects dynamic and context memories into selected layers of the frozen LLM through dual-source gated cross-attention. Experiments on urban and physical-dynamics benchmarks validate the framework and its key components, demonstrating competitive forecasting performance, improved robustness under sparse observations, strong generalization to unseen regions, and substantially reduced token usage and inference latency compared with existing LLM-based forecasters.

AI USE STATEMENT

Generative AI tools were used to assist with language polishing, formatting, and improving the readability of the manuscript. All AI-assisted content was reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

Inci M Baytas, Cao Xiao, Xi Zhang, Fei Wang, Anil K Jain, and Jiayu Zhou. Patient subtyping via time-aware lstm networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 65–74, 2017.

Marin Biloš, Johanna Sommer, Syama Sundar Rangapuram, Tim Januschowski, and Stephan Günnemann. Neural flows: Efficient alternative to neural odes. *Advances in neural information processing systems*, 34:21325–21337, 2021.

Defu Cao, Furong Jia, Sercan O. Arik, Tomas Pfister, Yixiang Zheng, Wen Ye, and Yan Liu. TEMPO: Prompt-based generative pre-trained transformer for time series forecasting, April 2024.

Ching Chang, Wei-Yao Wang, Wen-Chih Peng, and Tien-Fu Chen. LLM4TS: Aligning pre-trained LLMs as data-efficient time-series forecasters. *ACM Trans. Intell. Syst. Technol.*, 16(3):1–20, June 2025. ISSN 2157-6904, 2157-6912. doi: 10.1145/3719207.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Xiaodan Chen, Xiucheng Li, Bo Liu, and Zhijun Li. Biased temporal convolution graph network for time series forecasting with missing values. In *The Twelfth International Conference on Learning Representations*, 2023.

Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. *Advances in neural information processing systems*, 32, 2019.

Zheng Dong, Renhe Jiang, Haotian Gao, Hangchen Liu, Jinliang Deng, Qingsong Wen, and Xuan Song. Heterogeneity-informed meta-parameter learning for spatiotemporal time series forecasting. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 631–641, 2024.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. *Advances in neural information processing systems*, 32, 2019.

Yuxiao Hu, Qian Li, Dongxiao Zhang, Jinyue Yan, and Yuntian Chen. Context-Alignment: Activating and Enhancing LLM Capabilities in Time Series, April 2025.

Zijie Huang, Yizhou Sun, and Wei Wang. Learning continuous system dynamics from irregularly-sampled partial observations. *Advances in Neural Information Processing Systems*, 33:16177–16187, 2020.

Sheo Yon Jhin, Seojin Kim, and Noseong Park. Addressing prediction delays in time series forecasting: A continuous gru approach with derivative regularization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1234–1245, 2024.

Haoyu Jiang, Boan Qu, Junjie Zhu, Fanjie Zeng, Xiaojie Lin, and Wei Zhong. HyperLoad: A cross-modality enhanced large language model-based framework for green data center cooling load prediction, December 2025.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, et al. Time-llm: Time series forecasting by reprogramming large language models. *arXiv preprint arXiv:2310.01728*, 2023a.

Yilun Jin, Kai Chen, and Qiang Yang. Transferable graph structure learning for graph-based traffic forecasting across cities. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 1032–1043, 2023b.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in neural information processing systems*, 33:6696–6707, 2020.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International conference on machine learning*, pp. 2688–2697. Pmlr, 2018.

Zekun Li, Shiyang Li, and Xifeng Yan. Time series as images: Vision transformer for irregularly sampled time series, 2023. URL https://arxiv.org/abs/2303.12799.

Zhonghang Li, Lianghao Xia, Jiabin Tang, Yong Xu, Lei Shi, Long Xia, Dawei Yin, and Chao Huang. Urbangpt: Spatio-temporal large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5351–5362, 2024a.

Zhonghang Li, Lianghao Xia, Jiabin Tang, Yong Xu, Lei Shi, Long Xia, Dawei Yin, and Chao Huang. Urbangpt: Spatio-temporal large language models, 2024b.

Zachary C Lipton, David Kale, and Randall Wetzel. Directly modeling missing data in sequences with rnns: Improved classification of clinical time series. In *Machine learning for healthcare conference*, pp. 253–270. PMLR, 2016.

Chenxi Liu, Qianxiong Xu, Hao Miao, Sun Yang, Lingzheng Zhang, Cheng Long, Ziyue Li, and Rui Zhao. Timecma: Towards llm-empowered multivariate time series forecasting via cross-modality alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 18780–18788, 2025.

Yong Liu, Guo Qin, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. AutoTimes: Autoregressive time series forecasters via large language models. *Advances in Neural Information Processing Systems*, 37:122154–122184, December 2024. doi: 10.52202/079017-3882.

Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. *Advances in neural information processing systems*, 29, 2016.

Zijie Pan, Yushan Jiang, Sahil Garg, Anderson Schneider, Yuriy Nevmyvaka, and Dongjin Song. S^2IP-LLM: Semantic space informed prompt learning with LLM for time series forecasting. In *Forty-First International Conference on Machine Learning*, June 2024.

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Satya Narayan Shukla and Benjamin M Marlin. Multi-time attention networks for irregularly sampled time series. *arXiv preprint arXiv:2101.10318*, 2021.

Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 914–921, 2020.

Chenxi Sun, Hongyan Li, Yaliang Li, and Shenda Hong. TEST: Text prototype aligned embedding to activate LLM's ability for time series, February 2024.

Binwu Wang, Jiaming Ma, Pengkun Wang, Xu Wang, Yudong Zhang, Zhengyang Zhou, and Yang Wang. Stone: A spatio-temporal ood learning framework kills both spatial and temporal shifts. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2948–2959, 2024a.

Xinlei Wang, Maike Feng, Jing Qiu, Jinjin Gu, and Junhua Zhao. From news to forecast: Integrating event analysis in llm-based time series forecasting with reflection. *Advances in Neural Information Processing Systems*, 37:58118–58153, 2024b.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. *arXiv preprint arXiv:1906.00121*, 2019.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 753–763, 2020.

Hao Xue and Flora D Salim. Promptcast: A new prompt-based learning paradigm for time series forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 36(11):6851–6864, 2023.

Junchen Ye, Zihan Liu, Bowen Du, Leilei Sun, Weimiao Li, Yanjie Fu, and Hui Xiong. Learning the evolutionary and multi-scale graph structure for multivariate time series forecasting. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 2296–2306, 2022.

Kun Yi, Qi Zhang, Wei Fan, Hui He, Liang Hu, Pengyang Wang, Ning An, Longbing Cao, and Zhendong Niu. Fouriergnn: Rethinking multivariate time series forecasting from a pure graph perspective, 2023. URL https://arxiv.org/abs/2311.06190.

Rose Yu, Yaguang Li, Cyrus Shahabi, Ugur Demiryurek, and Yan Liu. Deep learning: A generic approach for extreme condition traffic forecasting. In *Proceedings of the 2017 SIAM international Conference on Data Mining*, pp. 777–785. SIAM, 2017.

Yuan Yuan, Jingtao Ding, Jie Feng, Depeng Jin, and Yong Li. Unist: A prompt-empowered universal model for urban spatio-temporal prediction. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4095–4106, 2024.

Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Weijia Zhang, Chenlong Yin, Hao Liu, Xiaofang Zhou, and Hui Xiong. Irregular multivariate time series forecasting: A transformable patching graph neural networks approach. In *Forty-first International Conference on Machine Learning*, 2024.

Weijia Zhang, Chenlong Yin, Hao Liu, and Hui Xiong. Unleashing the power of pre-trained language models for irregularly sampled time series. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 3831–3842, 2025.

Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems*, 21(9):3848–3858, 2019.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Tian Zhou, Peisong Niu, Liang Sun, Rong Jin, et al. One fits all: Power general time series analysis by pretrained lm. *Advances in neural information processing systems*, 36:43322–43355, 2023.

A APPENDIX

A.1 DATASET DETAILS

Urban Datasets (NYC) We evaluate on three New York City spatio-temporal datasets: **NYCtaxi**, **NYCbike**, and **NYCcrime**, which capture taxi flows, bike-share flows, and crime incidents, respectively. Following common practice in urban computing, we partition the city into grid-like regions using latitude and longitude coordinates. NYCtaxi contains 263 regions at approximately $3\,\mathrm{km} \times 3\,\mathrm{km}$ resolution, while NYCbike and NYCcrime contain 2162 regions at approximately $1\,\mathrm{km} \times 1\,\mathrm{km}$ resolution. To better reflect real-world urban data characteristics, we formulate all three datasets as *irregularly sampled* sequences.

EVENT-LEVEL SOURCE RECORDS. The original NYC datasets are collected in an event-level format, where each row records an individual event rather than a value aggregated over a predefined temporal grid. For example, in NYCtaxi, each row corresponds to a single taxi trip. Table 4 shows one representative record. The `vendor_id` identifies the data provider, while `pickup_datetime` and `dropoff_datetime` indicate the start and end times of the trip. The `passenger_count` records the number of passengers, and the pickup and drop-off longitude–latitude pairs specify the origin and destination of the event. Therefore, the raw dataset consists of asynchronously occurring trips with event-specific timestamps and spatial locations, rather than regularly sampled regional time series.

Table 4: An example of an event-level taxi-trip record.

vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
1	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982154846191406	40.767936706542969	-73.964630126953125	40.765602111816406

To construct the spatio-temporal forecasting benchmarks, we first assign individual events to spatial regions according to their coordinates and then aggregate them into region-level signals. For NYCtaxi and NYCbike, pickup and drop-off events are converted into regional inflow and outflow counts, while NYCcrime events are aggregated by region and crime category. In this way, the original event stream is transformed into graph-structured node observations, where each node corresponds to an urban region.

IRREGULAR SAMPLING CONSTRUCTION. Unlike standard time-series preprocessing that places all observations on a uniformly spaced temporal grid, we retain non-uniform observation intervals when constructing the input sequences. Consequently, consecutive observations can be separated by different elapsed times, and different spatial nodes may contain asynchronous or incomplete histories. For NYCtaxi and NYCbike, the resulting sampling intervals range from 10 to 360 minutes, with an average interval of approximately 60 minutes. For NYCcrime, the intervals range from 1 to 16 days, with an average interval of approximately 4 days. This construction preserves the event-driven and irregular characteristics of the original data and provides a realistic setting for evaluating forecasting models under non-uniform timestamps and missing observations.

TASK SETUP. We define the observations and forecasting horizons as follows. For taxi and bike datasets, the observation at time t is defined as the cumulative flow from 00:00 of the same day up to time t. For crime data, we use a sliding time window to compute a sequence of cumulative crime counts within the window as observations. We adopt forecasting horizons of using the past 24 hours to predict the next 12 hours for taxi/bike datasets, and using the past 96 days to predict the next 48 days for crime data.

TRAINING AND EVALUATION PROTOCOLS. Following the experimental protocol of UrbanGPT Li et al. (2024a), during the instruction-tuning stage, we randomly select 80 regions from the three NYC datasets as training data. The region indices are kept consistent for NYCbike and NYCcrime datasets. We use the following time spans for instruction tuning: for **NYCtaxi**, Jan. 1, 2017 to Mar. 31, 2017; for **NYCbike**, Apr. 1, 2017 to Jun. 30, 2017; and for **NYCcrime**, Jan. 1, 2016 to Dec. 31, 2018. For pretraining the spatio-temporal dependency encoder and training baseline models, we use the same training data and set the maximum number of epochs to 100.

In the testing stage, we conduct two evaluation settings:

1. **Supervised prediction:** We evaluate the model on a long continuous interval using all data from Dec. 2021 for NYCtaxi and NYCbike, and all data from the entire year of 2021 for NYCcrime.
2. **Zero-shot prediction:** We use an additional 80 regions from the NYC datasets as unseen test regions. For NYCtaxi and NYCbike, we use the first two weeks of data in 2020 for testing. For NYCcrime, we use the entire year of 2020 for testing.

NORMALIZATION. We normalize each NYC dataset independently using statistics computed from its *training split* only, and apply the same transformation to validation/test data to avoid information leakage. Specifically, for each variable we perform z-score normalization:

$$\tilde{x} = \frac{x - \mu_{\text{train}}}{\sigma_{\text{train}}}, \tag{12}$$

where μ_{train} and σ_{train} denote the mean and standard deviation estimated on the training split.

Physics-Based Simulation Datasets We additionally evaluate our model on two physics-based simulated datasets, namely **Springs** and **Charged**, following Kipf et al. Kipf et al. (2018). Each sample contains five interacting particles moving in a 2D box without external forces, where particles may collide with the box boundaries but do not experience friction or damping. Using a standard physics simulator, we generate trajectories of 60,000 integration steps for Springs and 12,000 steps for Charged.

IRREGULAR SAMPLING CONSTRUCTION. To simulate realistic sensing scenarios with limited and asynchronous observations, we construct irregularly sampled partial observations as follows. First, we reduce the temporal resolution by subsampling the simulated trajectories using different downsampling factors: 500 for the Springs system and 100 for the charged system, reflecting their different characteristic timescales. Then, for each particle independently, we sample the number of observations $n \sim \mathcal{U}(20, 28)$ and select n observation timestamps uniformly from the subsampled time grid. This procedure creates asynchronous observations where different particles are observed at different times, mimicking real-world sensor networks.

TASK SETUP. We formulate the forecasting task on the subsampled trajectories. For both systems, the first 60 sampled time steps define the history window, from which $n \sim \mathcal{U}(20, 28)$ timestamps are retained as irregular observations. The subsequent 60 sampled time steps are used as prediction targets. Due to different downsampling factors, this corresponds to different numbers of original integration steps: 30,000 steps for the Springs system (downsampling factor 500) and 6,000 steps for the charged particles system (downsampling factor 100). This extrapolation setting evaluates the model's ability to forecast long-term dynamics beyond the observation window.

NORMALIZATION. We compute the min–max statistics *only* on the training split and apply the same transformation to validation/test data to avoid information leakage. Specifically, we linearly rescale each state variable to the range $[-1, 1]$:

$$\tilde{x} = 2 \cdot \frac{x - x_{\min}^{\text{train}}}{x_{\max}^{\text{train}} - x_{\min}^{\text{train}}} - 1, \tag{13}$$

where $x_{\min}^{\text{train}}$ and $x_{\max}^{\text{train}}$ denote the minimum and maximum values of the corresponding state variable over the training split.

A.2 TRAINING CONFIGURATION

LLMODE is trained with a two-stage strategy to separately optimize the continuous dynamic representation learning and the frozen-LLM forecasting objective.

Stage 1: Training the Graph-Aware ODE Encoder. In the first stage, we pretrain the graph-aware ODE encoder using the GRU-ODE-Bayes parameterization De Brouwer et al. (2019), which jointly

considers the reconstruction likelihood of observed features and the Bayesian posterior consistency regularization:

$$\mathcal{L}_{\mathrm{ODE}} = \sum_{k} \left(\mathcal{L}_{\mathrm{pre}}^{(k)} + \beta_{\mathrm{post}} \mathcal{L}_{\mathrm{post}}^{(k)} \right), \tag{14}$$

where $\mathcal{L}_{\mathrm{pre}}^{(k)}$ and $\mathcal{L}_{\mathrm{post}}^{(k)}$ denote the pre-update reconstruction loss and the post-update posterior regularization loss at observation step k, respectively, and β_{post} controls the strength of the posterior regularization. Both losses are computed only on observed dimensions to accommodate feature-wise missing observations.

After this stage, the graph-aware ODE encoder generates continuous-time latent trajectories $\mathbf{H}_{\mathrm{dyn}}$ from irregular observations, which are subsequently distilled into dynamic memory tokens for the subsequent forecasting stage.

Stage 2: Training the Frozen-LLM Forecasting Framework. In the second stage, we optimize the forecasting framework while keeping the pretrained LLM backbone and the graph-aware ODE encoder frozen. Specifically, the trainable parameters include the dynamic memory resampler, the language encoder and context-memory resampler, dual-source gated cross-attention modules, and the forecasting regression head. The parameters of the LLM backbone and the graph-aware ODE encoder are not updated during training.

The final objective minimizes the prediction error between the forecasts and ground-truth states at all query timestamps:

$$\mathcal{L}_{\mathrm{forecast}} = \frac{1}{\sum_v |Q_v|} \sum_{v} \sum_{t \in Q_v} \|\hat{\mathbf{y}}_v(t) - \mathbf{y}_v(t)\|_2^2, \tag{15}$$

where Q_v denotes the query timestamps of node v, $\hat{\mathbf{y}}_v(t), \mathbf{y}_v(t) \in \mathbb{R}^D$ denote the predicted and ground-truth multivariate target vectors, respectively.

A.3 Implementation Configuration

Model configuration. The latent dimension of the graph-aware ODE encoder is set to $d_{\mathrm{ode}} = 50$. The ODE states and text-encoder outputs are separately projected into the memory space of dimension d_{mem} before dynamic- and context-memory resampling, respectively. For Vicuna-7B, the LLM hidden dimension is $d_{\mathrm{LLM}} = 4096$. Both memory streams are compressed using a two-layer Perceiver Resampler with eight attention heads and a per-head dimension of 64. The fixed token budget is set to $K = 5$ for both the dynamic and context memories. The strength coefficient of the dynamics-aware attention bias is set to $\beta = 0.1$, and the mixing coefficient λ is set to 0.5 to balance the contributions of the two memories. For Vicuna-7B, the dual-source gated cross-attention modules are inserted after Transformer Layers 1 and 16. The maximum prompt length is set to 2048 tokens.

Optimization and hardware. The trainable modules are optimized using AdamW with a learning rate of 2×10^{-4}, cosine learning-rate scheduling, and a warmup ratio of 0.03. We use a per-device batch size of 32 and apply early stopping according to the validation loss with a patience of 10 epochs. All experiments are repeated using five random seeds, i.e., 42, 43, 44, 45, and 46, and the average results are reported. All results are obtained from the best validation checkpoint. Training is performed in BF16 precision on NVIDIA A100 GPUs.

A.4 GRU-ODE Parameterization and Continuous Propagation

For each node v, the latent state evolves according to the Neural ODE in Eq. 3. Following the GRU-ODE formulation De Brouwer et al. (2019), we instantiate the vector field as

$$\frac{d\mathbf{h}_v(t)}{dt} = f_\theta(\mathbf{h}_v(t)) = \big(1 - \mathbf{z}_v(t)\big) \odot \big(\mathbf{g}_v(t) - \mathbf{h}_v(t)\big), \tag{16}$$

where $\mathbf{z}_v(t)$ is the update gate, $\mathbf{g}_v(t)$ is the candidate state, and $\odot$ denotes element-wise multiplication. Both quantities are computed from the current latent state through learnable projections included in θ. This parameterization gives the continuous vector field a gated update structure analogous to a discrete GRU while preserving continuous-time state evolution.

Let t_{i-1} and t_i be two consecutive observation times of node v. The pre-update state at t_i is obtained by numerical integration:

$$\mathbf{h}_v(t_i^-) = \mathrm{ODESolve}\big(f_\theta, \bar{\mathbf{h}}_v(t_{i-1}^+), [t_{i-1}, t_i]\big)\,. \tag{17}$$

Here, $\mathrm{ODESolve}(\cdot)$ denotes a numerical ODE solver that integrates the vector field f_θ from the initial state $\bar{\mathbf{h}}_v(t_{i-1}^+)$ over the interval $[t_{i-1}, t_i]$, yielding the propagated state $\mathbf{h}_v(t_i^-)$ immediately before the observation at t_i is incorporated. We use the Euler method as the default solver due to its simplicity and computational efficiency. Because the integration interval explicitly uses $t_i - t_{i-1}$, the encoder naturally handles nonuniform sampling gaps and can be queried at arbitrary intermediate timestamps. The graph-corrected state $\bar{\mathbf{h}}_v(t_{i-1}^+)$ is used as the initial condition for the next interval.

A.5 Observation Encoding and State Correction

At time t_i, $\mathbf{h}_v(t_i^-)$ represents the propagated latent state before the new observation $\mathbf{x}_{v,i}$ is incorporated. The role of Φ is to convert this observation into a correction signal for updating the propagated state. To do so, Φ first estimates what the observation should be according to the current latent state, and then compares this estimate with the actually observed value.

Under a Gaussian observation model, we use a learnable observation head Γ_ω to predict the feature-wise mean and scale from the propagated latent state:

$$[\boldsymbol{\mu}_{v,i}; \boldsymbol{\sigma}_{v,i}] = \Gamma_\omega\big(\mathbf{h}_v(t_i^-)\big)\,, \qquad \Gamma_\omega : \mathbb{R}^{d_{\mathrm{ode}}} \to \mathbb{R}^{2D}, \tag{18}$$

where Γ_ω is a lightweight neural network parameterized by ω. It maps the d_{ode}-dimensional latent state to the predicted mean $\boldsymbol{\mu}_{v,i} \in \mathbb{R}^D$ and scale $\boldsymbol{\sigma}_{v,i} \in \mathbb{R}^D$ of the D input features. The predicted mean represents the observation expected from the current latent state, while the predicted scale characterizes the corresponding feature-wise uncertainty.

For the r-th feature, we compute the normalized discrepancy between the new observation and its predicted value:

$$\delta_{v,i,r} = \frac{x_{v,i,r} - \mu_{v,i,r}}{\sigma_{v,i,r} + \epsilon}, \qquad r = 1, \ldots, D, \tag{19}$$

where ϵ is a small constant for numerical stability. A large magnitude of $\delta_{v,i,r}$ indicates that the newly observed value differs substantially from the value expected by the propagated latent state.

We then combine the predicted observation, the actual observation, and their discrepancy into a feature-wise descriptor:

$$\mathbf{s}_{v,i,r} = [\mu_{v,i,r}; \sigma_{v,i,r}; x_{v,i,r}; \delta_{v,i,r}]\,. \tag{20}$$

Each descriptor is mapped to a feature embedding:

$$\mathbf{u}_{v,i,r} = \rho(\mathbf{A}_r \mathbf{s}_{v,i,r} + \mathbf{b}_r)\,, \tag{21}$$

where $\mathbf{A}_r$ and $\mathbf{b}_r$ are learnable parameters for the r-th feature, and $\rho(\cdot)$ is a nonlinear activation function.

Finally, the feature embeddings are concatenated to obtain the complete observation encoding:

$$\Phi\big(\mathbf{x}_{v,i}, \mathbf{h}_v(t_i^-)\big) = [\mathbf{u}_{v,i,1}; \ldots; \mathbf{u}_{v,i,D}]\,. \tag{22}$$

Therefore, Φ summarizes what the current latent state predicts, what is actually observed, and how much the two differ. The correction GRU combines this observation information with $\mathbf{h}_v(t_i^-)$ to obtain the corrected state $\mathbf{h}_v(t_i^+)$ in Eq. 4. This observation-driven correction anchors the continuously propagated state to newly arrived measurements and reduces accumulated propagation errors.

A.6 Graph-Aware State Correction

After the observation-driven jump, all node states at the current event time are aligned by propagating nodes without new observations to the same timestamp. We then use the graph structure to refine each node state with information from its neighboring nodes.

For each edge $(u, v) \in \mathcal{E}$, we compute a relative message from node u to node v:

$$\begin{aligned}\mathbf{m}_{u,v} = \psi\big(\mathbf{h}_u(t_i^+) - \mathbf{h}_v(t_i^+)\big) \\ - \psi\big(\mathbf{h}_v(t_i^+) - \mathbf{h}_u(t_i^+)\big)\,,\end{aligned} \tag{23}$$

where $\psi(\cdot)$ is a lightweight MLP. The messages from all neighboring nodes are aggregated as

$$\mathbf{a}_v = \sum_{u \in \mathcal{N}(v)} \mathbf{m}_{u,v}, \tag{24}$$

where $\mathcal{N}(v)$ denotes the set of neighboring nodes of node v, and $\mathbf{a}_v$ summarizes the structural information received by node v.

A node-specific gate is then computed to control how much neighboring information is incorporated:

$$\boldsymbol{\eta}_v = \sigma\big(\mathbf{W}_\eta[\mathbf{h}_v(t_i^+); \mathbf{a}_v] + \mathbf{b}_\eta\big), \tag{25}$$

where $[\cdot;\cdot]$ denotes concatenation and $\sigma(\cdot)$ is the sigmoid function. The graph-aware state is obtained through a residual correction:

$$\bar{\mathbf{h}}_v(t_i^+) = \mathbf{h}_v(t_i^+) + \gamma\, \boldsymbol{\eta}_v \odot \mathbf{a}_v, \tag{26}$$

where $\odot$ denotes element-wise multiplication and γ controls the overall correction strength.

The relative messages characterize the state differences between connected nodes, while the gate adaptively determines how strongly the aggregated neighboring information modifies the current node state. The resulting $\bar{\mathbf{h}}_v(t_i^+)$ is used as the initial state for the next continuous propagation interval and is also recorded to construct the latent trajectory $\mathbf{H}_{\mathrm{dyn}}$.

A.7 Perceiver-Based Fixed-Budget Resampling

For clarity, we omit the node index v in this subsection and describe the resampling for a single-node trajectory. The Perceiver Resampler maintains K learnable latent queries $\{\mathbf{q}_k\}_{k=1}^K$, where $\mathbf{q}_k \in \mathbb{R}^{d_{\mathrm{mem}}}$. For each latent state $\mathbf{h}(t_i)$, the corresponding key and value vectors are computed as

$$\mathbf{k}_i = \mathbf{W}_K \mathbf{h}(t_i), \qquad \mathbf{v}_i = \mathbf{W}_V \mathbf{h}(t_i), \tag{27}$$

where $\mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d_{\mathrm{mem}} \times d_{\mathrm{ode}}}$ are learnable mappings, and $\mathbf{k}_i, \mathbf{v}_i \in \mathbb{R}^{d_{\mathrm{mem}}}$.

To encourage the resampler to preserve states from rapidly changing intervals, we directly introduce the magnitude of the instantaneous latent change into the attention logits:

$$\alpha_{k,i} = \mathrm{softmax}_i\left(\frac{\mathbf{q}_k^\top \mathbf{k}_i}{\sqrt{d_{\mathrm{mem}}}} + \beta\ \mathrm{Norm}_{1:L}(\|f_\theta(\mathbf{h}(t_i))\|_2)\right). \tag{28}$$

Here, $\alpha_{k,i}$ is the attention weight from the k-th latent query to the state at timestamp t_i; $f_\theta(\mathbf{h}(t_i)) = \dot{\mathbf{h}}(t_i)$ is the instantaneous latent change determined by the ODE vector field; $\mathrm{Norm}_{1:L}(\cdot)$ denotes normalization over the L timestamps of the current trajectory; and β controls the strength of the dynamics-aware bias.

The representation produced by the k-th latent query is obtained by aggregating the value vectors:

$$\mathbf{z}_k = \sum_{i=1}^{L} \alpha_{k,i} \mathbf{v}_i, \qquad \mathbf{z}_k \in \mathbb{R}^{d_{\mathrm{mem}}}. \tag{29}$$

Because K is fixed independently of the original trajectory length L, the resampler converts trajectories of different lengths into a fixed number of dynamic memory tokens.

A.8 Explicit Time Labeling

Each resampled vector $\mathbf{z}_k$ aggregates information from multiple timestamps. To indicate the temporal location of the selected dynamic content, we compute an attention-weighted center time using the same attention weights:

$$t_k^{\mathrm{center}} = \sum_{i=1}^{L} \alpha_{k,i} t_i. \tag{30}$$

The center time represents the temporal position around which the information selected by the k-th latent query is concentrated.

We first encode t_k^{center} using multi-frequency sinusoidal features:

$$\boldsymbol{\phi}(t) = [\sin(\omega_1 t), \cos(\omega_1 t), \ldots, \sin(\omega_{M_\tau} t), \cos(\omega_{M_\tau} t)]. \tag{31}$$

The sinusoidal features are then transformed into a d_{mem}-dimensional time representation:

$$\boldsymbol{\tau}_k = \Gamma_\tau\big(\boldsymbol{\phi}(t_k^{\text{center}})\big) \in \mathbb{R}^{d_{\text{mem}}}, \tag{32}$$

where Γ_τ is a lightweight learnable network for time encoding.

The time representation is fused with the resampled dynamic content through element-wise addition:

$$\tilde{\mathbf{z}}_k = \mathbf{z}_k + \boldsymbol{\tau}_k, \qquad \tilde{\mathbf{z}}_k \in \mathbb{R}^{d_{\text{mem}}}. \tag{33}$$

Finally, the K time-aware dynamic memory tokens are stacked as

$$\tilde{\mathbf{Z}} = [\tilde{\mathbf{z}}_1; \ldots; \tilde{\mathbf{z}}_K] \in \mathbb{R}^{K \times d_{\text{mem}}}. \tag{34}$$

The resulting $\tilde{\mathbf{Z}}$ serves as the dynamic memory for subsequent alignment and interaction with the query-slot hidden states in the frozen LLM.

A.9 Construction of Context Memory

Descriptor format. We first summarize the historical observations into segment-wise statistics and serialize them into a compact, structured textual description of the form

```
[SEG_STATS S=S] seg1: ... segS: ... max_gap=...
[/SEG_STATS]
```

where S is the number of segments used to partition the history window.

Meaning of fields. For each segment `segj`, we report: (i) `*_mean`: the mean value of the corresponding variable within the segment (computed over observed points); (ii) `[max: *, min: *]`: the maximum and minimum values within the segment (over observed points); (iii) `obs_rate`: the observation rate in the segment, i.e., the fraction of timestamps in the segment that are observed. In addition, `max_gap` denotes the time gap between the last observed timestamp in the history window and the first query time to be predicted.

After obtaining the textual description, we encode it into context token representations, i.e., contextualized token representations using a pretrained text encoder:

$$\mathbf{E}_P = [\mathbf{e}_1; \ldots; \mathbf{e}_{L'_p}] \in \mathbb{R}^{L'_p \times d_{\text{mem}}}, \tag{35}$$

where L'_p is the number of contextualized token representations, and d_{mem} is the memory-token dimension after projecting the text-encoder outputs. We then apply the context-memory resampler, which uses the same latent-query resampling mechanism introduced in Sec. 3.2, without the dynamics-aware attention bias and explicit time labeling, to compress the variable-length token representations into a fixed number of context memory tokens:

$$\mathbf{P} = [\mathbf{p}_1; \ldots; \mathbf{p}_K] \in \mathbb{R}^{K \times d_{\text{mem}}}, \tag{36}$$

where K is the fixed number of context memory tokens.

Example: Springs.

```
[SEG_STATS S=4]
seg1: pos_mean=[x: -0.05, y: -0.10], x=[max: 0.11, min: -0.16],
y=[max: 0.04, min: -0.19], obs_rate=60.00%
seg2: pos_mean=[x: 0.16, y: -0.72], x=[max: 0.35, min: 0.05], y=[max:
-0.47, min: -0.85], obs_rate=20.00%
seg3: pos_mean=[x: 0.31, y: -0.74], x=[max: 0.52, min: 0.06], y=[max:
-0.67, min: -0.83], obs_rate=20.00%
seg4: pos_mean=[x: 0.33, y: -0.65], x=[max: 0.42, min: 0.24], y=[max:
-0.35, min: -0.98], obs_rate=40.00% max_gap=0.00
[/SEG_STATS]
```

Here, `pos_mean` summarizes the average 2D position (x, y) in each segment, while `x=[max,min]` and `y=[max,min]` capture the range of motion along each axis. The `obs_rate` indicates how densely the trajectory is observed within the segment, and `max_gap` measures the gap from the last observation to the first query time.

Example: NYCtaxi.

```
[SEG_STATS S=4]
seg1: inflow_mean=-0.34, outflow_mean=-0.26, inflow=[max: -0.34, min:
-0.34], outflow=[max: -0.26, min: -0.26], obs_rate=83.33%
seg2: inflow_mean=-0.29, outflow_mean=-0.26, inflow=[max: -0.27, min:
-0.32], outflow=[max: -0.25, min: -0.26], obs_rate=100.00%
seg3: inflow_mean=-0.23, outflow_mean=-0.25, inflow=[max: -0.18, min:
-0.26], outflow=[max: -0.25, min: -0.25], obs_rate=50.00%
seg4: inflow_mean=-0.08, outflow_mean=-0.24, inflow=[max: -0.07, min:
-0.11], outflow=[max: -0.24, min: -0.24], obs_rate=83.33% max_gap=5.00
[/SEG_STATS]
```

This snippet summarizes the segment-wise statistics of `inflow` and `outflow` for a region. The `*_mean` fields describe the average level within each segment, the `[max,min]` fields capture within-segment variation, `obs_rate` reflects missingness, and `max_gap` indicates the forecast gap from the last observed record to the first query slot.

A.10 Prompt Examples

Below, we provide representative prompt examples for the urban and physical-dynamics datasets.

NYCtaxi region prompt.

Task: Predict future inflow and outflow at the query slots given historical observations over NYC regions, where each node corresponds to a region described by its POI profile.
Region: Bronx area with dominant POIs: Cultural Facility, Residential, and Education Facility.
Query-Slots: ¡q_1¿ ¡q_2¿ $\cdots$ ¡q_{L_q}¿.

Charged interacting-particle prompt.

Task: Given irregularly sampled observations of the particle system, predict each particle's future 2D position (x, y) at the query slots.
System: A 2D interacting-particle system governed by pairwise forces.
Query-Slots: ¡q_1¿ ¡q_2¿ $\cdots$ ¡q_{L_q}¿.

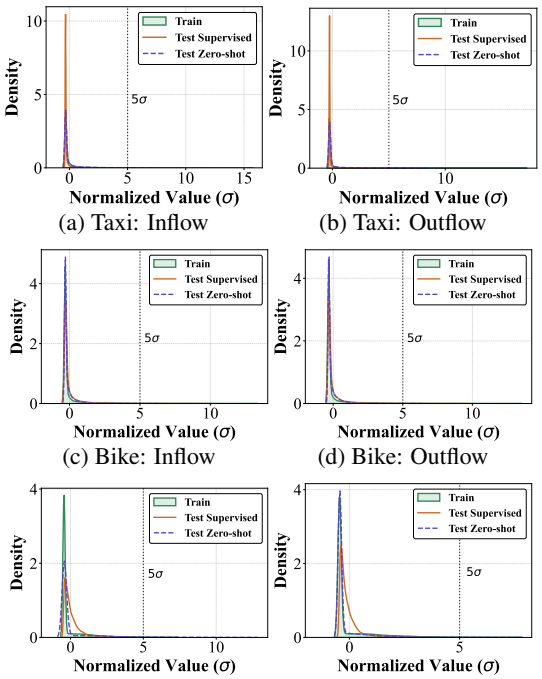
Spring-coupled particle prompt.

Task: Given irregularly sampled observations of the particle system, predict each particle's future 2D position (x, y) at the query slots.
System: A 2D particle system with spring-based interactions.
Query-Slots: ¡q_1¿ ¡q_2¿ $\cdots$ ¡q_{L_q}¿.

Each query-slot token <q$_i$> corresponds to an irregular query timestamp $t_i^{(q)}$ and represents a time-specific prediction request. For example, when predicting traffic flow at 10:01, 11:34, and 15:47, the query slots <q_1>, <q_2>, and <q_3> are instantiated as <10:01>, <11:34>, and <15:47>, respectively. The temporal information is encoded using the same time-embedding scheme introduced in Appendix A.8.

A.11 Distribution Shift Analysis of NYC Datasets

In this work, we use z-score normalized data for both training and testing on the NYC datasets, where normalization is performed using the mean and standard deviation computed from the training split. From Figure 9, we observe that all three datasets exhibit long-tail distributions. To avoid the impact of these long-tail distributions on model performance in irregular forecasting and ensure a



(a) Taxi: Inflow (b) Taxi: Outflow

(c) Bike: Inflow (d) Bike: Outflow

(e) Crime: robbery (f) Crime: burglary

Figure 9: Normalized distribution analysis across datasets. The vertical dashed line indicates the 5σ threshold used for data filtering.

fair comparison across methods, we remove points with values greater than $+5\sigma$, resulting in an approximately 2.2% data removal rate. Importantly, this filtering is performed once during dataset preprocessing, and the resulting processed data are used identically for all baseline methods and LLMODE. Therefore, no method receives method-specific outlier treatment.

Although the same preprocessing protocol is applied to all datasets and methods, differences remain between the training and test distributions. We therefore further analyze two representative cases: the relatively large errors observed on NYCcrime and the comparatively small performance gains on NYCtaxi inflow.

Distribution Shift in NYCcrime. Despite this normalization and outlier removal, we find that the test distributions for the NYCcrime dataset still exhibit a mismatch compared to the training split. This discrepancy persists due to differences in the data distribution between the training and test sets. Since all models are trained and evaluated on the normalized data, this mismatch can disproportionately affect methods sensitive to scale or extreme values. In such cases, a small number of large errors can be amplified under squared-error metrics like MSE, leading to significantly worse performance on the NYCcrime dataset.

Distribution Shift in NYCtaxi Inflow. The distribution shift of NYCtaxi inflow is less visually apparent in Figure 9a; however, its summary statistics reveal a substantial contraction from the training split to the supervised test split. After normalization using the training-split statistics, the standard deviation decreases from 1.000 to 0.321, while the 95th percentile decreases from 1.569 to 0.195. These statistics indicate that the supervised test period has a considerably narrower and more stable value range than the training period. Under such a concentrated evaluation distribution, dominant temporal patterns are relatively stable, and the advantage of explicitly reconstructing continuous-time dynamics may be less pronounced in terms of MAE. As a result, interval-aware but token-discrete methods such as ISTS-PLM can remain competitive on this specific variable. Nevertheless, LLMODE still achieves a slightly lower MSE on NYCtaxi inflow, suggesting better suppression of relatively large prediction errors. Together with the stronger improvements observed on NYCbike, NYCcrime, and the physical-system benchmarks, we therefore regard the NYCtaxi inflow result as a dataset-specific case arising from its concentrated test distribution.

Table 5: Sensitivity to the solver step size on the Springs dataset.

Configuration	MAE	MSE
Best baseline	0.1053	0.0237
Δt_0	**0.0973**	**0.0191**
$2\Delta t_0$	0.1039	0.0207

Table 6: Sensitivity to the choice of ODE solver on the Springs dataset.

Solver	MAE	MSE	Latency (ms/sample)
Euler	**0.0973**	**0.0191**	**0.5854**
RK4	0.0986	0.0197	1.6876

A.12 Numerical Configuration Sensitivity

LLMODE formulates the evolution between observations as differential equations and obtains continuous states through numerical integration. To verify that its performance is not tied to a particular numerical configuration, we further analyze the effects of the solver step size and solver choice on the Springs-X forecasting task.

Sensitivity to Solver Step Size. The default solver step size Δt_0 is set to the minimum sampling interval. As shown in Table 5, increasing the step size to $2\Delta t_0$ slightly increases the MAE and MSE from $0.0973/0.0191$ to $0.1039/0.0207$. This is expected because a finer integration grid provides a more accurate approximation of the continuous trajectory. Nevertheless, LLMODE with the larger step size still outperforms the best baseline, demonstrating its robustness to the choice of solver step size.

Sensitivity to ODE Solver. We further replace the default Euler solver with the higher-order RK4 solver. As reported in Table 6, RK4 provides no accuracy improvement but increases the ODE-solving latency from 0.5854 to 1.6876 ms per sample. This indicates that Euler already provides sufficient numerical accuracy when the step size matches the minimum sampling interval, while offering substantially lower computational cost. Together, these results show that LLMODE is not dependent on a specific solver or step-size configuration.

A.13 Detailed Token-Budget and Efficiency Results

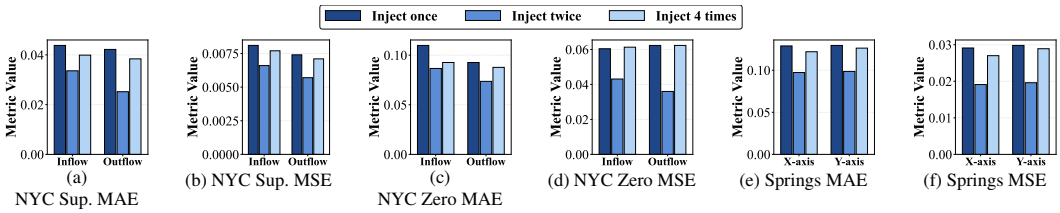
Table 7 provides the detailed numerical results corresponding to the token-budget efficiency analysis in Figure 4, including token consumption and inference latency under different LLM-based forecasting settings. Compared with UrbanGPT, LLMODE reduces the inference latency from 646.73 ms to approximately 84 ms while substantially lowering the token budget, thereby reducing the computational overhead within the frozen-LLM forecasting paradigm.

The reported token budget includes prompt tokens and both dynamic and context memory tokens. Unlike the appended spatio-temporal tokens used by UrbanGPT, the ODE-derived tokens are maintained as external memory and injected through gated cross-attention, without increasing the length of the LLM self-attention sequence. Compared with full-trajectory exposure, fixed-budget compression reduces the reported token budget by approximately 75% while maintaining or improving forecasting accuracy. Nevertheless, the compressed and full-trajectory settings exhibit relatively similar inference latency, since changing the external-memory size has a more limited computational impact than changing the LLM input-sequence length.

Our objective is to explore a new modeling paradigm for irregular spatio-temporal forecasting by leveraging the representation capacity, prior knowledge, and transferability of pretrained LLMs. Accordingly, the efficiency analysis focuses on reducing the cost of processing external spatio-temporal evidence, rather than making an absolute speed comparison with all non-LLM methods. Nevertheless, we additionally report the inference latency of representative non-LLM forecasting methods as a computational reference. Their substantially lower latency reveals the remaining efficiency gap

Table 7: Token-budget and efficiency analysis on NYCbike zero-shot. Non-LLM forecasting methods are included for reference, while token-budget analysis focuses on LLM-based forecasting settings.

Variant	MAE	MSE	Latency (ms)	Tokens
LLM-based forecasting settings				
UrbanGPT	0.0778	0.0292	646.73	665
$K = 1$	0.0863	0.0402	83.29	64
$K = 3$	0.0826	0.0301	83.49	68
$K = 5$	**0.0745**	0.0260	83.81	72
$K = 8$	0.0748	**0.0254**	84.10	78
$K = 30$	0.0782	0.0288	85.48	122
$K = 90$	0.0831	0.0322	86.04	242
w/o compression	0.0839	0.0331	86.54	302
Non-LLM forecasting methods				
MTGNN	0.0769	0.0357	9.87	–
tPatchGNN	0.0765	0.0323	10.63	–



(a) NYC Sup. MAE (b) NYC Sup. MSE (c) NYC Zero MAE (d) NYC Zero MSE (e) Springs MAE (f) Springs MSE

Figure 10: Performance under different injection frequencies. We use Vicuna-7B as the frozen backbone with 32 Transformer blocks and insert the dual-source gated injection modules between the blocks. We compare three configurations: one insertion after Block 1, two insertions with a 16-block interval, and four insertions with an 8-block interval. NYC Sup. and NYC Zero denote the supervised and zero-shot settings, respectively.

and motivates future work to further improve the computational efficiency of LLM-based forecasting while preserving the predictive and generalization benefits of pretrained language models.

A.14 Additional Analysis of Gated Evidence Injection

Insertion Frequency and Number. The frozen Vicuna-7B backbone contains 32 Transformer blocks. We insert the dual-source gated injection modules between these blocks and jointly vary their insertion frequency and number. Specifically, we consider three configurations: (i) one early injection at Block 1, (ii) two injections with an interval of 16 blocks, and (iii) four injections with an interval of 8 blocks. Figure 10 reports the results on NYCtaxi-Supervised, NYCtaxi-Zero-shot, and Springs.

As shown in Figure 10, inserting two modules every 16 blocks achieves the best overall performance and consistently improves over using only one early injection. Increasing the insertion frequency to every 8 blocks does not yield further gains and can even degrade accuracy. This suggests that overly frequent injection may introduce redundant updates into the frozen representations. Moreover, each additional module requires extra cross-attention computation at another depth of the LLM, increasing both computational cost and inference latency. We therefore adopt two gated injection modules with a 16-block interval as the default configuration.

Gate Evolution and Effective Contribution. Under the default every-16-blocks configuration, two gated injection modules are inserted into the frozen Vicuna-7B backbone. Layer 0 and Layer 1 in Figure 7 denote the first and second injection modules, respectively. For each module, the learnable gate parameter g is transformed into the effective gate gain $\alpha = \tanh(g)$. The solid curves in Figure 7 show the evolution of α throughout training.

Because the gate gain alone does not fully characterize the actual magnitude of the injected update, we additionally report the normalized Effective Contribution:

$$\mathcal{C} = \frac{\alpha \|h\|_2}{\|h_{\text{prompt}}\|_2}, \tag{37}$$

where h denotes the representation produced by the corresponding cross-attention branch, and h_{prompt} denotes the prompt-token representation before the gated residual update. The metric therefore measures the magnitude of the injected update relative to the current backbone representation. The shaded regions in Figure 7 visualize this normalized contribution.

Starting from zero initialization, the gate gains of both modules gradually increase and eventually stabilize. This behavior shows that the model progressively opens the injection paths, incorporates the external dynamic and contextual tokens, and finally converges to stable injection strengths. Although Layer 0 generally learns a smaller gate gain than Layer 1, the gate values cannot be compared in isolation because the actual update also depends on the magnitude of the attended representation h. The Effective Contribution provides this complementary view and confirms that both injection modules maintain meaningful non-zero updates after convergence.

A.15 Qualitative Case Study

Figure 8 presents a representative forecasting example on NYCtaxi under the supervised setting, comparing LLMODE with the strong irregular-aware baseline tPatchGNN Zhang et al. (2024). For each evaluation window, the models use a 24-hour history to predict the following 12 hours. For visualization, forecasts from consecutive windows are arranged chronologically to form a 124-hour trajectory.

Across the evaluation period, LLMODE follows the ground-truth trends more consistently and exhibits smaller deviations than tPatchGNN. The enlarged region around hours 58–64 further illustrates their difference during a sharp demand variation. LLMODE more accurately captures both the magnitude and temporal location of the peak, whereas tPatchGNN exhibits larger amplitude errors and temporal misalignment. Similar behavior is observed for both inflow and outflow, indicating that the improvement is not limited to a single forecasting variable. Overall, the case study qualitatively demonstrates that LLMODE better preserves complex temporal patterns and produces more stable forecasts under irregular spatio-temporal observations.

A.16 Baseline Details

To ensure fair comparisons across methods with different temporal assumptions, we apply a unified adaptation for baselines that only operate on a *regular* time grid. On the input side, we linearly interpolate the irregular observations along time to obtain regularly sampled sequences before feeding them into such models. On the evaluation side, the protocol differs across datasets. For the NYC datasets, the observations are event-driven and do not provide an underlying complete regularly sampled series from which a missingness mask can be defined, so we standardize the target as predicting the next 12 *regular* time steps and evaluate all methods on these aligned horizons. For the physics simulation datasets, the original trajectories are generated and recorded on a fixed-step grid, and we create irregular inputs by subsampling this complete regular sequence and specifying irregular query times as prediction targets. This construction provides explicit indices indicating which future regular steps correspond to the query times. Therefore, for methods that can only produce regular-step forecasts, we ask them to predict the next 60 regular steps and then select the predictions at the indexed query-aligned steps for evaluation, enabling consistent assessment under irregular query horizons.

We compare our method with a diverse set of baselines, including pre-trained language model based approaches and graph neural network based methods, covering both irregularly sampled and regularly sampled spatio-temporal forecasting settings.

TGCN Zhao et al. (2019) is a temporal graph convolutional network for spatio-temporal forecasting, which combines graph convolution to model spatial dependencies with recurrent units to capture temporal dynamics, and is originally designed for regularly sampled data defined over a fixed graph.

STSGCN Song et al. (2020) is a spatial-temporal graph convolutional network for spatio-temporal forecasting, which jointly models spatial and temporal dependencies through localized spatial-temporal subgraphs. It is originally designed for regularly sampled spatio-temporal data defined over a fixed graph.

MTGNN Wu et al. (2020) utilizes a learnable graph structure to model multivariate temporal correlations. MTGNN employs 1-D dilation convolutions to generate temporal representations.

BiTGraph Chen et al. (2023) is a biased temporal convolution based graph network for time series forecasting with missing values, which combines graph neural networks to model spatial dependencies with biased temporal convolutions to handle irregular sampling patterns and missing observations.

FourierGNN Yi et al. (2023) transforms multivariate sequences into the frequency domain to model spatio-temporal dependencies via spectral graph convolutions.

ISTS-PLM Zhang et al. (2025) is a pre-trained language model based approach for irregularly sampled time series forecasting, designed to handle non-uniform time gaps and missing observations through LLM-compatible series representations.

UrbanGPT Li et al. (2024b) integrates a spatio-temporal dependency encoder with a Large Language Model to capture complex urban dynamics via spatio-temporal instruction tuning.

GPT4TS Zhou et al. (2023) reframes time series forecasting as a language modeling task by processing patch-based representations through a frozen pre-trained GPT-2 backbone.

GRU-ODE De Brouwer et al. (2019) combines Gated Recurrent Units with Neural Ordinary Differential Equations to model continuous-time latent states for irregularly sampled time series.

tPatchGNN Zhang et al. (2024) is a graph neural network based approach for modeling irregularly sampled time series, which represents observations as temporal patches and captures temporal dependencies through patch-level message passing to handle non-uniform time gaps and missing observations.

ViTST Li et al. (2023) is a vision transformer based model for irregularly sampled time series analysis, which transforms irregular multivariate time series into line graph images and leverages vision transformers for representation learning and forecasting.