EVIL Baseline: LLM-Discovered Heuristics can be strong Baselines for Scientific Inference

Anonymous Authors[1]

Abstract

We explore whether LLM-guided evolutionary search can automatically discover simple, interpretable algorithms that serve as strong baselines for scientific inference. Using `EVIL` (**EV**olving **I**nterpretable algorithms with **L**LMs), we evolve compact Python/NumPy programs that perform zero-shot inference on dynamical systems. Across three scientific inference tasks (temporal point process prediction, Markov jump process rate estimation, and time series imputation), the LLM-discovered heuristics are surprisingly competitive with state-of-the-art deep learning models, while being orders of magnitude faster, fully interpretable, and discoverable in minutes for under $1 of API cost. These results provide a humbling reminder that the added complexity of sophisticated models is not always necessary, and suggest that LLM-guided program search can serve as a useful tool for establishing strong baselines that help calibrate where complex approaches genuinely add value.

1. Introduction

Deep learning has driven impressive advances in scientific inference, from neural temporal point processes (Mei & Eisner, 2017; Mei et al., 2022; Zeng et al., 2024) to Markov jump process estimators (Seifner & Sánchez, 2023; Berghaus et al., 2024) and diffusion-based time series models (Tashiro et al., 2021; Seifner et al., 2025). A natural question when evaluating such methods is how much of their performance comes from learned representations versus from the structure already present in the data. Strong baselines help answer this question, but designing good baselines by hand is difficult: domain experts may not explore the full space of possible heuristics, and ad-hoc baselines (e.g., last-value imputation, majority-class prediction) are often too simple to be informative.

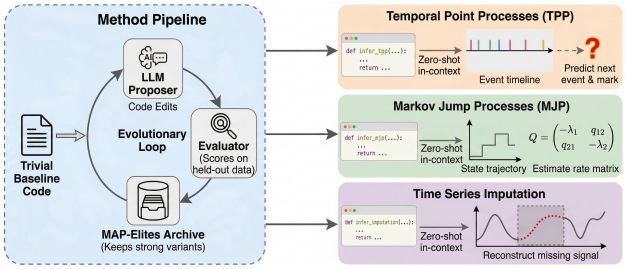

Figure 1. Overview of the `EVIL` approach. LLM-guided evolutionary search discovers one interpretable Python/NumPy inference function per task that generalizes across datasets in a zero-shot manner, providing a strong automatically generated baseline.

We propose using AI itself to help close this gap: rather than hand-crafting baselines, we use LLM-guided evolutionary search to *automatically discover* strong heuristic baselines.

Our approach, `EVIL` (**EV**olving **I**nterpretable algorithms with **L**LMs), leverages AlphaEvolve-style program evolution (Novikov et al., 2025) to search over short Python/NumPy programs that perform zero-shot, in-context inference. The result is a single compact, readable function per task that generalizes across datasets without per-dataset training. We apply this to three scientific inference problems: temporal point process prediction, Markov jump process rate estimation, and time series imputation.

Perhaps surprisingly, across all three domains the LLM-discovered heuristics are competitive with state-of-the-art deep learning models, and outperform them on several benchmarks (Sections 4–6). These heuristics evaluate $\sim$200$\times$ faster on commodity CPUs than neural baselines on GPUs, are fully interpretable as readable Python functions, and cost under $1 and 15 minutes of wall-clock time to discover. They also generalize to unseen datasets with far more categories than encountered during evolution (Table 3) and show robustness to distribution shifts (Figure 2).

Our main contributions are:

- **A new role for LLM-guided program evolution in science**: We demonstrate that AlphaEvolve-style

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

search (Novikov et al., 2025) can discover general-purpose scientific inference functions, not just solve individual problem instances. This represents a form of autonomous scientific algorithm discovery.

- **Humbling baselines for scientific inference**: The LLM-discovered heuristics are surprisingly competitive with deep learning models across three dynamical systems tasks, raising the bar for what a "baseline" can achieve and sharpening the case that must be made for more complex approaches.
- **A practical tool for practitioners**: `EVIL` provides a cheap, fast, and reproducible way to establish strong baselines before committing to complex model development.

We hypothesize that these heuristics perform well because (i) the LLM brings extensive prior knowledge about statistical methods, (ii) evolved programs perform exact computations on context data rather than approximating through learned parameters, and (iii) the short-program constraint imposes an Occam's-razor-like bias toward generalizable solutions (Georgiev et al., 2025). [1]

2. Related Work

LLM-guided program evolution. AlphaEvolve (Novikov et al., 2025) introduced an evolutionary framework where LLMs propose code modifications scored by automated evaluators, achieving advances in algorithm design and mathematical discovery (Georgiev et al., 2025). FunSearch (Romera-Paredes et al., 2023) pioneered a similar approach for combinatorial problems, and LLM-SR (Shojaee et al., 2024) applies LLM-based search to symbolic regression. Our work extends this paradigm from single-instance optimization to discovering general-purpose scientific inference functions.

Foundation models for scientific inference. Foundation Inference Models (FIMs) use synthetic pretraining for zero-shot estimation across dynamical systems, including FIM-MJP (Berghaus et al., 2024), FIM-PP (Berghaus et al., 2026), and FIM-ℓ (Seifner et al., 2025). Prior-fitted networks (Müller et al., 2022) pursue a similar amortized strategy. These form the main baselines in our experiments; domain-specific methods are introduced in Sections 4–6. Very recently, Zhang & Gilpin (2026) proposed *context parroting* (which is a simple baseline that copies relevant patterns from the input context) and show that it is surprisingly competitive with foundation models in chaotic time series forecasting, reinforcing the message that simple, interpretable baselines can challenge trained models.

[1] Our source code is available at `https://anonymous.4open.science/r/evil_algorithms-374D/`.

3. Methodology

3.1. Problem Setup

Our goal is to use LLM-guided search to automatically discover a strong heuristic baseline for a given scientific inference task. In each of our three applications, the task can be framed as: given context observations and a target query, produce a prediction using a pure Python/NumPy function. For point processes, this means predicting the next event given a history and context sequences. For MJPs, it means estimating rate matrices from discrete observations. For imputation, it means filling in missing values given partially observed time series. In all cases, the function must work across datasets without modification, operating in a zero-shot, in-context manner at test time.

3.2. Evolutionary Search

We use OpenEvolve (Sharma, 2025), an open-source implementation of AlphaEvolve (Novikov et al., 2025), to evolve our inference functions. The evolution starts from a trivial initial program and iteratively improves it over the iterations. At each iteration, an LLM proposes code modifications in diff format, the modified program is evaluated against a scoring function, and promising solutions are stored in an evolutionary database. The database implements MAP-Elites (Mouret & Clune, 2015) with island-based populations to balance exploration and exploitation. We use an ensemble of LLMs for code generation, with the system prompt constraining solutions to use only NumPy (no deep learning frameworks). Full hyperparameters and system prompts are provided in Appendix G. We kept the initial programs, system prompts, and scoring functions fixed throughout the study and did not optimize them based on benchmark results to avoid implicit overfitting.

3.3. Evaluation Protocol

To prevent overfitting to test data, we create our own evaluation subsplit from the existing training split of each dataset. This means `EVIL` has access to strictly less data than baseline methods that train on the full training set. The scoring function for evolution measures prediction quality on this held-out subsplit (e.g., accuracy and RMSE for point processes, cross-entropy and RMSE for MJPs, MAE for imputation). We report two variants: `EVIL`, evolved on subsplits of the target datasets, and `EVIL (synthetic prior)`, evolved on synthetic data only.

To clarify what we mean by "zero-shot," the final evolved program is applied at test time without any per-dataset training or parameter fitting. For `EVIL`, the target-dataset subsplits are used only during evolution to guide the search toward a good general-purpose algorithm; the resulting program itself contains no dataset-specific parameters and

generalizes to entirely unseen datasets (e.g., MIMIC-II, Table 3). `EVIL (synthetic prior)` provides the strictest zero-shot baseline, evolved entirely on synthetic data, to isolate whether an LLM-evolved heuristic can generalize from a broad synthetic prior to real-world datasets.

We keep the main experimental setup fixed to 100 LLM iterations, with 20% of the suggestions coming from GPT-5 and 80% from GPT-5-mini. Additional ablations on dataset-specific evolution, variance across independent runs, and a substantially longer run are reported in Appendix C.

4. Temporal Point Processes

4.1. Background

A marked temporal point process (MTPP) (Daley & Vere-Jones, 2003) is a stochastic process generating sequences of events $(t_i, k_i)_{i=1}^{N}$, where each event has a timestamp $t_i \in \mathbb{R}^+$ and a categorical mark $k_i \in \{1, \dots, K\}$. The process is characterized by its conditional intensity function $\lambda^*(t, k)$, governing the instantaneous rate of events given the history. The central prediction task is forecasting the next N events given a history of past events. Appendix A gives a slightly more detailed formal description, summarizes the datasets, and briefly reviews the point-process baselines.

4.2. Experimental Setup

We evolve a function that takes target histories and a pool of context sequences, and returns predicted next-event times and marks. The initial program simply predicts the global median inter-event gap and the majority mark type. The evolutions (100 iterations) took about 15 minutes wall-clock time. We evolve two variants: `EVIL`, which was evolved on small amounts of subdata from the target datasets in Table 1, and `EVIL (synthetic prior)`, which was evolved only on subsplits of the synthetic data from (Berghaus et al., 2026). This setup makes the comparison to FIM-PP (Berghaus et al., 2026) particularly natural: like `EVIL (synthetic prior)`, FIM-PP is built around synthetic pretraining and is then evaluated either zero-shot or after finetuning on real data. At the same time, FIM-PP was pretrained on substantially more data overall (14.4 million events, whereas `EVIL (synthetic prior)` was evolved on only 72k).

Baselines. We compare against strong point-process baselines including A-NHP, NHP, IFTPP, Dual-TPP, CDiff, HawkesEM, and FIM-PP (Mei & Eisner, 2017; Mei et al., 2022; Shchur et al., 2020; Deshpande et al., 2021; Zeng et al., 2024; Lewis & Mohler, 2011; Berghaus et al., 2026). These include neural architectures trained separately on each dataset, a classical Hawkes-process baseline, and a pretrained foundation model; all are substantially more complex than our final heuristic, and all except the pretrained FIM-PP variants require per-dataset training.

Table 1. Long-horizon prediction (N=5) on five real-world datasets. We show OTD and sMAPE$_{\Delta t}$ (both lower is better) for the best neural baseline (CDiff), FIM-PP variants, and `EVIL` variants. Full results including all baselines are in Appendix B. Baseline results from (Zeng et al., 2024; Berghaus et al., 2026). FIM(zs) is FIM-PP (Berghaus et al., 2026) in zero-shot mode, FIM(f) is FIM-PP after finetuning on the target dataset. EVIL(s) is `EVIL (synthetic prior)`.

	Taxi		Taobao		StackOv.		Amazon		Retweet	
Method	OTD	sM	OTD	sM	OTD	sM	OTD	sM	OTD	sM
CDiff	5.97	89.5	10.15	124.3	10.74	100.6	**9.48**	81.3	15.86	106.6
HawkesEM	7.15	86.4	11.27	164.1	11.70	**82.8**	14.64	173.7	15.69	80.5
FIM(zs)	6.77	74.9	15.95	168.3	11.52	93.3	11.12	119.1	15.75	98.7
FIM(f)	4.08	71.1	13.17	146.9	**10.35**	86.4	10.03	78.7	15.65	83.0
EVIL(s)	4.39	**69.7**	**9.89**	**121.3**	11.91	86.4	11.46	**57.9**	15.58	92.3
EVIL	**4.05**	71.0	10.94	166.1	11.65	84.4	10.93	71.2	**15.56**	**82.5**

Table 2. Runtime for one N=5 point-process evaluation. Other neural baselines are slower overall because they must also be trained per dataset.

Method	Hardware	Time
FIM-PP(zs)	A100 40GB	964s (16min)
`EVIL`	1 CPU core	5s

Table 3. Generalization to MIMIC-II (Lee et al., 2011), an unseen dataset with 75 marks (vs. max 22 during evolution). Baselines from (Panos, 2024).

Method	RMSE	Accuracy
THP (Zuo et al., 2021)	1.00	84.88
SAHP (Zhang et al., 2020)	1.49	83.61
A-NHP (Mei et al., 2022)	1.00	84.81
IFTPP (Shchur et al., 2020)	0.74	85.13
DTPP (Panos, 2024)	**0.72**	85.51
`EVIL (synthetic prior)`	0.90	83.14
`EVIL`	0.89	**87.80**

4.3. Discovered Algorithm

The evolved point-process heuristic (see Pseudocode 1 and Python Algorithm 1 in Appendix F) can be summarized as: *(1) from the context sequences, estimate a smoothed mark-transition table and typical inter-event gaps, including mark-specific average gaps; (2) predict the next time by mixing recent gaps with the average gap associated with the last mark; (3) predict the next mark by combining local transition counts from the prefix with the global transition table.*

4.4. Results and Analysis

Table 1 summarizes long-horizon (N=5) prediction across five datasets. The LLM-discovered heuristic is surprisingly competitive with the best neural methods on most dataset–metric combinations, despite being a simple NumPy program. The practical speedup is also notable: FIM-PP evaluation took 964s (about 16 minutes) on an A100 40GB GPU, whereas `EVIL` took 5s on a single CPU core, a $\sim$200$\times$ speedup on much simpler hardware (Table 2). The other neural baselines have even higher end-to-end cost because they must first be trained for each dataset.

Results for N=10 and N=20 follow similar trends and are reported in Appendix B.

Table 4. Next-event prediction (N=1) on two real-world datasets. Baseline results from (Zeng et al., 2024; Berghaus et al., 2026). Best in bold.

	TAXI			TAOBAO		
Method	$\text{RMSE}_{\Delta t}$	Acc	$\text{sMAPE}_{\Delta t}$	$\text{RMSE}_{\Delta t}$	Acc	$\text{sMAPE}_{\Delta t}$
A-NHP	0.32	**0.91**	85.13	0.53	0.47	129.13
Dual-TPP	0.34	**0.91**	89.12	0.53	0.47	131.43
NHP	0.34	**0.91**	90.63	0.53	0.46	133.69
IFTPP	0.38	0.90	90.03	0.53	0.45	126.01
CDiff	0.34	**0.91**	87.12	0.52	0.48	127.12
HawkesEM	0.34	0.09	79.93	**0.13**	0.56	111.57
FIM-PP (zs)	0.34	0.52	98.03	**0.13**	0.52	112.89
FIM-PP (f)	0.45	**0.91**	85.91	0.16	0.60	115.98
EVIL (synthetic prior)	0.30	**0.91**	**69.49**	**0.13**	**0.61**	**95.71**
EVIL	**0.29**	**0.91**	71.98	**0.13**	0.59	117.62

Generalization. Table 4 shows next-event (N=1) prediction results on TAXI and TAOBAO. On TAXI, both EVIL variants match the best neural baselines in accuracy (0.91) while substantially improving sMAPE. The TAXI dataset is special in the sense that it often contains alternations between marks (see Figure 4 in the appendix). This is not covered well by the synthetic training data of (Berghaus et al., 2026), which is why FIM-PP(zs) performs comparatively poorly. Notably, EVIL (synthetic prior) however already achieves 91% accuracy on TAXI using only synthetic training data, whereas the non-finetuend FIM-PP(zs) reaches only 52%. This indicates that the LLM-evolved algorithm was able to generalize to special patterns that were not part of the synthetic training data.

Moreover, Table 3 demonstrates the generalization capabilities of EVIL on MIMIC-II (Lee et al., 2011), a medical event dataset with 75 mark types that was *never seen during evolution* (the maximum during evolution was 22 marks since we evolved on the datasets of Table 1 or synthetic data). EVIL achieves 87.8% accuracy, surpassing all neural baselines that were specifically trained on this dataset. This is particularly striking because neural TPP models typically have a fixed output dimension matching the training mark count and cannot easily accommodate new marks, whereas the EVIL algorithm naturally handles arbitrary mark counts through its use of transition statistics computed at inference time.

5. Markov Jump Processes

5.1. Background

A Markov jump process (MJP) (Norris, 1998) is a continuous-time stochastic process taking values in a finite state space $\{1, \dots, K\}$. It is fully characterized by a rate matrix $Q \in \mathbb{R}^{K \times K}$ with non-negative off-diagonal entries and rows summing to zero, together with an initial distribution π_0. The fundamental inference task is to estimate Q and π_0 from noisy, discretely observed trajectories. This is a challenging inverse problem because exact jump times are unobserved.

Table 5. DFR rate matrix parameters inferred by different methods, expressed as ratios to the ground truth (1.0 is perfect). V: potential strength, r/b: rate parameters.

	V	r	b
Ground Truth	1.00	1.00	1.00
NeuralMJP	1.06	1.17	1.14
FIM-MJP	1.11	**0.99**	**0.98**
EVIL (synthetic prior)	**0.97**	1.04	1.06

MJP inference arises naturally in several scientific domains. We evaluate on four datasets that span different areas of science: **(i)** the *Discrete Flashing Ratchet* (DFR) (Ajdari & Prost, 1992), a 6-state model from non-equilibrium statistical physics that describes a Brownian particle driven by an asymmetric, periodically switching potential; accurately inferring its generator enables the computation of entropy production, a key thermodynamic quantity; **(ii)** *Ion Channel* (IonCh) recordings from the viral potassium channel $\text{Kcv}_{\text{MT325}}$ (Gazzarrini et al., 2006), where the MJP describes switching between discrete conductance states and is used to characterize channel gating kinetics; **(iii)** *Alanine Dipeptide* (ADP) (Mironov et al., 2019), a canonical molecular dynamics benchmark where the MJP captures conformational transitions between metastable states of a small peptide, relevant for understanding protein folding; and **(iv)** *Protein Folding* (PFold) (Mardt et al., 2017), a simpler 2-state system modeling folding–unfolding transitions. In all these settings, recovering an accurate rate matrix is not just a prediction task but yields direct scientific insight into the kinetics and thermodynamics of the underlying system. Appendix D provides additional background on MJPs, the datasets, the evaluation metrics, and the DFR entropy-production calculation.

5.2. Experimental Setup

We evolve a function that takes discrete observation grids and state sequences and returns a rate matrix and initial distribution. The initial program computes simple normalized transition counts. The evolutions (100 iterations) took about 14 minutes wall-clock time. Since ground-truth rate matrices are only available for synthetic data, EVIL (synthetic prior) is evolved using 7,200 synthetic MJPs from the same family as FIM-MJP (Berghaus et al., 2024), with state spaces of size 2–6.

Baselines. We compare against NeuralMJP (Seifner & Sánchez, 2023), a neural network trained separately on each target dataset, and FIM-MJP (Berghaus et al., 2024), a foundation model pretrained on 45K synthetic MJPs. Both are substantially more complex architectures with millions of parameters.

Table 6. Time-averaged Hellinger distances ($\times 10^{-2}$, lower is better) between empirical and simulated processes. Mean (std.) over 100 histograms.

Model	DFR	IonCh	ADP	PFold
NeuralMJP	0.30 (0.06)	0.48 (0.02)	1.38 (0.52)	**0.015** (**0.015**)
FIM-MJP	0.27 (0.06)	0.41 (0.02)	1.39 (0.47)	**0.014** (**0.014**)
`EVIL (synthetic prior)`	**0.16** (0.02)	**0.30** (0.06)	**0.93** (0.23)	**0.015** (**0.013**)

5.3. Discovered Algorithm

The evolved MJP heuristic (see Pseudocode 3 and Python Algorithm 3 in Appendix F) is: *(1) estimate the initial distribution from smoothed counts of the first observations; (2) for each state, accumulate how long trajectories stay there and how often they leave; (3) set the exit rate to approximately exits divided by exposure time, with simple clipping and noise filtering; (4) distribute this exit rate across destination states using recency-weighted transition counts with smoothing.*

5.4. Results and Analysis

We begin with the Discrete Flashing Ratchet (DFR) (Ajdari & Prost, 1992), a physically meaningful 6-state non-equilibrium system where accurate recovery of the generator has direct scientific value (e.g., for computing entropy production). Table 5 shows that the LLM-discovered heuristic follows the DFR structure well and is competitive with FIM-MJP, with the clearest advantage on the potential strength V that controls the ratchet's asymmetry.

Figure 2 then evaluates a more demanding downstream quantity on the same benchmark: entropy production, which is nonzero in the DFR because the ratchet sustains irreversible probability currents. The qualitative picture is again favorable to `EVIL (synthetic prior)`, which stays closer to the ground truth than FIM-MJP across the shown range. At high voltages the DFR becomes intrinsically harder because a few transitions dominate while others become extremely rare: some rates differ by roughly four orders of magnitude, so the rare transitions are only weakly represented in the observed paths. That makes entropy production sensitive to estimation noise in those low-count entries, but `EVIL (synthetic prior)` still recovers the very rarely observed transitions more precisely than FIM-MJP.

Table 6 then broadens the picture to all four datasets. For the real-world systems (ion-channel gating, IonCh; molecular conformational dynamics, ADP and PFold), the recovered generator is the primary scientific output, since it encodes the kinetic rates that govern the system's behavior. We use the Hellinger distance (Hellinger, 1909) between empirical and model-simulated state distributions to assess whether the inferred generators produce realistic trajectories, even when ground-truth rate matrices are unavailable.

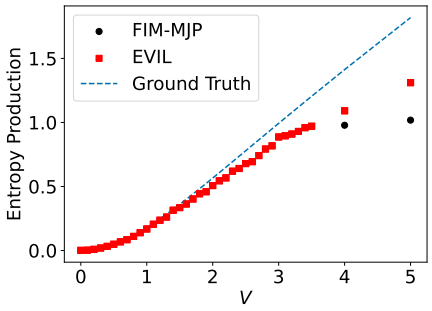

Figure 2. Entropy production on the DFR. This serves as a stress test for when inference becomes harder as the rates become increasingly dissimilar. Across the shown potentials, `EVIL (synthetic prior)` tracks the ground truth more closely than FIM-MJP.

Appendix D briefly explains the metric and its use. Interestingly, while NeuralMJP and FIM-MJP achieve very similar Hellinger distances to each other, `EVIL (synthetic prior)` is consistently better on DFR, IonCh, and ADP.

6. Time Series Imputation

6.1. Background

Time series imputation aims to reconstruct missing values in partially observed multivariate time series. In many applications, these observed sequences are not arbitrary collections of numbers, but partial observations of an underlying trajectory generated by a dynamical system. From this perspective, both observed and missing values lie on a latent time-evolving process, so imputation can be viewed as inferring the trajectory that best explains the available observations and then reading off the unobserved values. We consider two settings: point-wise missing patterns (random individual values are missing) and window-based missing patterns (contiguous blocks of observations are absent).

6.2. Experimental Setup

We evolve a function that takes observation values (with NaN at missing positions), timestamps, and a prediction mask, and returns fully imputed values. The initial program uses simple last-value prediction. The evolutions use 6 datasets: 4 with 50% point-wise missing patterns (Beijing, Italy, GuangZhou, PeMS) and 2 Motion Capture variants with 20% windowed missing patterns. The remaining 4 point-wise datasets (Pedestrian, Solar, ETT_h1, Electricity) are held out entirely and used only for zero-shot generalization evaluation. Appendix E summarizes these datasets and their sources. The evolutions (100 iterations) took about 37 minutes wall-clock time.

Table 7. MAE on 8 datasets with 50% point-wise missing. EVIL was evolved on the first four datasets (Beij., Italy, GZ, PeMS); the remaining four (Ped., Solar, ETT, Elec.) are zero-shot generalization targets. Baselines from (Fang et al., 2023; Du et al., 2024; Seifner et al., 2025). Best in bold.

	Air quality		*Traffic*			*Electricity*		
Method	Beij.	Italy	GZ	PeMS	Ped.	Solar	ETT	Elec.
BRITS	.169	.321	3.34	**.287**	.259	1.99	.238	1.12
SAITS	.194	.285	3.39	.302	**.205**	1.83	**.223**	1.40
GP-VAE	.258	.453	3.42	.346	.451	1.81	.414	1.10
CSDI	**.144**	.958	3.20	.288	.351	.804	.318	.798
BayOTIDE	–	–	2.69	–	–	.734	–	–
FIM-ℓ	.166	.215	2.43	.365	.273	.595	.279	.083
EVIL	.152	**.204**	**2.09**	.338	.242	**.413**	.267	**.078**

Table 8. MAE on Motion Capture (20% windowed missing). Baselines from (Seifner et al., 2025).

Model	MC (PCA)	MC (No PCA)
LatentODE	$\mathbf{1.66 \pm 0.99}$	–
Spline	3.36 ± 1.18	4.21 ± 1.44
Spline(F)	2.90 ± 0.87	3.00 ± 0.88
FIM-ℓ	1.77 ± 0.63	$\mathbf{1.61 \pm 0.45}$
EVIL	2.98 ± 2.04	3.34 ± 2.07

Baselines. We compare against BRITS (Cao et al., 2018) (bidirectional recurrent), SAITS (Du et al., 2023) (self-attention), GP-VAE (Fortuin et al., 2020) (Gaussian process VAE), CSDI (Tashiro et al., 2021) (conditional score-based diffusion), BayOTIDE (Fang et al., 2023) (Bayesian decomposition), LatentODE (Rubanova et al., 2019) (neural ODE), and FIM-ℓ (Seifner et al., 2025) (foundation model pretrained on 2M+ ODE solutions). All baselines except FIM-ℓ require per-dataset training. For the Motion Capture window-imputation setting, we additionally report cubic spline interpolation and a Savitzky–Golay filtered spline baseline (Spline(F)) as included in (Seifner et al., 2025).

6.3. Discovered Algorithm

The evolved imputation heuristic (see Pseudocode 4 and Python Algorithm 4 in Appendix F) is: *(1) detect contiguous missing blocks; (2) if a gap is long, perform motif retrieval by searching earlier in the same series for a recurring local pattern whose preceding context matches the context before the gap, then copy the following pattern with a level shift; (3) otherwise, fill the gap by time-aware linear interpolation.*

6.4. Results and Analysis

Table 7 shows point-wise imputation results. The LLM-discovered heuristic performs competitively on all 8 datasets, including the 4 held-out datasets (Ped., Solar, ETT, Elec.) that were never seen during evolution, demonstrating zero-shot generalization. This is consistent with findings from TSI-Bench (Du et al., 2024), which reported that simple interpolation methods can perform surprisingly well on point-wise missing patterns, though here the heuristic was discovered automatically rather than hand-chosen.

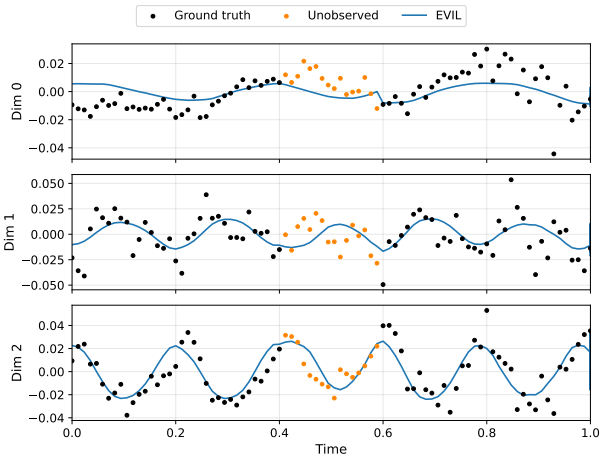

Figure 3. Imputation on Motion Capture. The motif retrieval strategy enables EVIL to predict non-trivial patterns in missing windows.

Table 8 shows windowed imputation on Motion Capture. Here, FIM-ℓ clearly outperforms EVIL, suggesting that reconstructing contiguous missing windows genuinely benefits from the learned dynamical priors that FIM-ℓ acquires during pretraining—a case where the gap between heuristic and neural model is unambiguous. At the same time, the variances are high across all methods, so this benchmark might be less conclusive. Still, EVIL discovers a simple nonlinear strategy based on *motif* retrieval (Lin, 2002): it searches earlier in the same sequence for a segment whose recent context resembles the context before the missing window, and then reuses the continuation that followed that earlier segment. This already allows EVIL to reconstruct structured oscillations in some cases (Figure 3). Interestingly, the concurrent *context parroting* baseline of Zhang & Gilpin (2026) also relies on a motif-based strategy—copying recurring patterns from the input context—suggesting that motif retrieval may be a naturally strong inductive bias for time series imputation that both human-designed and LLM-discovered methods converge on independently.[2]

7. Discussion: Implications for AI-Driven Science

Our results raise several interesting points.

LLM-discovered baselines as a calibration tool. We think EVIL-style baseline discovery deserves a place in the scientific inference workflow. Running a short evolutionary

[2]The context parroting paper has been released after the knowledge cutoff of the LLM, so the LLM seemingly discovered it independently.

search before committing to a complex model can reveal how much of a task's difficulty actually requires learned representations, as opposed to structure that is already accessible from the raw data. When the heuristic does well, the headroom left for more complex models is likely small. When it clearly falls short—as with windowed time series imputation (Table 8)—that is direct evidence that learned dynamical priors carry real weight, and the effort of training a richer model is warranted.

Autonomous algorithm discovery. `EVIL` shows one concrete way that LLMs can participate in scientific methodology beyond serving as writing or coding assistants. The evolved programs do not merely reproduce known algorithms; they combine familiar building blocks (transition tables, recency weighting, motif retrieval) in configurations we did not anticipate and could not find in the existing literature. The human role remains essential (choosing the task framing, designing the scoring function, and interpreting the result), but the algorithmic design itself is offloaded to the search.

Transparency and verification. Because every solution is a short, readable program, a domain scientist can trace each prediction back to concrete operations on the input data. This makes it straightforward to spot failure modes, sanity-check the logic against domain knowledge, or extend the heuristic by hand, none of which is easy with a black-box neural model.

8. Conclusion

We showed that LLM-guided evolutionary search can automatically discover compact, interpretable algorithms that serve as surprisingly strong baselines for scientific inference. Across temporal point processes, Markov jump processes, and time series imputation, these LLM-discovered heuristics are competitive with state-of-the-art deep learning models on many benchmarks, while being orders of magnitude faster, cheaper to discover, and fully transparent. We hope that automatically generated baselines of this kind can help researchers tell apart problems where deep learning is essential from those where a well-chosen heuristic already does most of the work.

9. Limitations

The main limitation of `EVIL` is its weaker performance on tasks that heavily benefit from dataset-specific learned representations, such as windowed time series imputation (Section 6). The discovered algorithms are also deterministic and cannot quantify uncertainty; extending LLM-guided evolution to discover stochastic inference procedures is an open direction. It is hard to assess the novelty of the discovered algorithms: while they seem intuitive in hindsight, they combine known concepts (transition tables, recency weighting, motif retrieval) with optimizations that would be tedious for humans to hand-tune. Finally, while the evolved algorithms generalize well across the datasets tested here, their performance on fundamentally different data distributions remains an open question.

References

Ajdari, A. and Prost, J. Mouvement induit par un potentiel périodique de basse symétrie: diélectrophorese pulsée. *Comptes rendus de l'Académie des sciences. Série 2, Mécanique, Physique, Chimie, Sciences de l'univers, Sciences de la Terre*, 315(13):1635–1639, 1992.

Bacry, E., Bompaire, M., Deegan, P., Gaïffas, S., and Poulsen, S. V. Tick: a python library for statistical learning, with an emphasis on hawkes processes and time-dependent models. *J. Mach. Learn. Res.*, 18(1): 7937–7941, January 2017. ISSN 1532-4435.

Berghaus, D., Cvejoski, K., Seifner, P., Ojeda, C., and Sanchez, R. J. Foundation inference models for markov jump processes. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=f4v7cmm5sC.

Berghaus, D., Seifner, P., Cvejoski, K., Ojeda, C., and Sanchez, R. J. In-context learning of temporal point processes with foundation inference models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=h9HwUAODFP.

Cao, W., Wang, D., Li, J., Zhou, H., Li, L., and Li, Y. BRITS: Bidirectional recurrent imputation for time series. *Advances in Neural Information Processing Systems*, 31, 2018.

Daley, D. J. and Vere-Jones, D. *An Introduction to the Theory of Point Processes*. Springer, 2003.

Deshpande, P., Marathe, K., De, A., and Sarawagi, S. Long horizon forecasting with temporal point processes. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pp. 571–579, 2021.

Du, W., Côté, D., and Liu, Y. SAITS: Self-attention-based imputation for time series. *Expert Systems with Applications*, 219:119619, 2023.

Du, W., Wang, J., Qian, L., Yang, Y., Ibrahim, Z., Liu, F., Wang, Z., Liu, H., Zhao, Z., Zhou, Y., et al. Tsi-bench: Benchmarking time series imputation. *arXiv preprint arXiv:2406.12747*, 2024.

Fang, S., Wen, Q., Luo, Y., Zhe, S., and Sun, L. Bayotide: Bayesian online multivariate time series imputation with functional decomposition. *arXiv preprint arXiv:2308.14906*, 2023.

Fortuin, V., Baranchuk, D., Rätsch, G., and Mandt, S. GP-VAE: Deep probabilistic multivariate time series imputation. In *Proceedings of AISTATS*, pp. 1651–1661, 2020.

Gazzarrini, S., Kang, M., Epimashko, S., Van Etten, J. L., Dainty, J., Thiel, G., and Moroni, A. Chlorella virus MT325 encodes water and potassium channels that interact synergistically. *Proceedings of the National Academy of Sciences*, 103(14):5355–5360, 2006.

Georgiev, B., Gómez-Serrano, J., Tao, T., and Wagner, A. Z. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025.

Gillespie, D. T. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

Hawkes, A. G. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

Heinonen, M., Yildiz, C., Mannerström, H., Intosalmi, J., and Lähdesmäki, H. Learning unknown ode models with gaussian processes. In *International Conference on Machine Learning*, pp. 1959–1968. PMLR, 2018.

Hellinger, E. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 136:210–271, 1909.

Husic, B. E., Charron, N. E., Lemm, D., Wang, J., Pérez, A., Majewski, M., Krämer, A., Chen, Y., Olsson, S., de Fabritiis, G., Noé, F., and Clementi, C. Coarse graining molecular dynamics with graph neural networks. *The Journal of Chemical Physics*, 153(19):194101, 2020.

Lee, J., Scott, D. J., Villarroel, M., Clifford, G. D., Saeed, M., and Mark, R. G. Open-access mimic-ii database for intensive care research. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 8315–8318. IEEE, 2011.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.

Lewis, E. and Mohler, G. A nonparametric em algorithm for multiscale hawkes processes. *Journal of Nonparametric Statistics*, 01 2011.

Lin, H., Wu, L., Zhao, G., Pai, L., and Li, S. Z. Exploring generative neural temporal point process. *Transactions on Machine Learning Research*, 2022.

Lin, J. Finding motifs in time series. In *Knowledge Discovery and Data Mining*, 2002. URL https://api.semanticscholar.org/CorpusID:8844326.

Mardt, A., Pasquali, L., Wu, H., and Noé, F. Vampnets for deep learning of molecular kinetics. *Nature Communications*, 9, 2017.

Mei, H. and Eisner, J. M. The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Mei, H., Qin, G., and Eisner, J. Imputing missing events in continuous-time event streams. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4475–4485. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/mei19a.html.

Mei, H., Yang, C., and Eisner, J. Transformer embeddings of irregularly spaced events and their participants. In *International Conference on Learning Representations*, 2022.

Mironov, V., Alexeev, Y., Mulligan, V. K., and Fedorov, D. G. A systematic study of minima in alanine dipeptide. *Journal of Computational Chemistry*, 40(2):297–309, 2019.

Mouret, J.-B. and Clune, J. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Müller, S., Hollmann, N., Arango, S. P., Grabocka, J., and Hutter, F. Transformers can do Bayesian inference. *arXiv preprint arXiv:2112.10510*, 2022.

Ni, J., Li, J., and McAuley, J. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 188–197, 2019. doi: 10.18653/v1/D19-1018.

Norris, J. R. *Markov Chains*. Cambridge University Press, 1998.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., and Balog, M. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Panos, A. Decomposable transformer point processes. *Advances in Neural Information Processing Systems*, 37: 88932–88955, 2024.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2023. doi: 10.1038/s41586-023-06924-6.

Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Seifert, U. Stochastic thermodynamics, fluctuation theorems and molecular machines. *Reports on Progress in Physics*, 75(12):126001, 2012.

Seifner, P. and Sánchez, R. J. Neural markov jump processes. In *International Conference on Machine Learning*, pp. 30523–30552. PMLR, 2023.

Seifner, P., Cvejoski, K., Körner, A., and Sanchez, R. J. Zero-shot imputation with foundation inference models for dynamical systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=NPSZ7V1CCY.

Sharma, A. Openevolve: an open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Shchur, O., Biloš, M., and Günnemann, S. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020.

Shojaee, P., Meidani, K., Farimani, A. B., and Reddy, C. K. LLM-SR: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

Tashiro, Y., Song, J., Song, Y., and Ermon, S. CSDI: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems*, 34:24804–24816, 2021.

Wang, J. M., Fleet, D. J., and Hertzmann, A. Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30 (2):283–298, 2007.

Xue, S., Shi, X., Zhang, J. Y., and Mei, H. HYPRO: A hybridly normalized probabilistic model for long-horizon prediction of event sequences. In *Advances in Neural Information Processing Systems*, 2022.

Yildiz, C., Heinonen, M., and Lahdesmaki, H. Ode2vae: Deep generative second order odes with bayesian neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Zeng, M., Regol, F., and Coates, M. Interacting diffusion processes for event sequence forecasting. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Zhang, Q., Lipani, A., Kirnap, O., and Yilmaz, E. Self-attentive Hawkes process. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 11183–11193. PMLR, 2020.

Zhang, Y. and Gilpin, W. Context parroting: A simple but tough-to-beat baseline for foundation models in scientific machine learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=EUAXc9Hlvm.

Zhou, K., Zha, H., and Song, L. Learning triggering kernels for multi-dimensional hawkes processes. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 1301–1309, 2013.

Zhu, H., Li, X., Zhang, P., Li, G., He, J., Li, H., and Gai, K. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1079–1088, 2018. doi: 10.1145/3219819.3219826.

Zuo, S., Jiang, H., Li, Z., Zhao, T., and Zha, H. Transformer hawkes process. In *International Conference on Machine Learning*, 2021.

A. Additional Temporal Point Process Details

A.1. Mathematical Setup

A marked temporal point process (MTPP) is a random sequence of events $\{(t_i, k_i)\}_{i=1}^n$ with strictly increasing times $t_i \in \mathbb{R}_+$ and discrete marks $k_i \in \{1, \dots, K\}$ (Daley & Vere-Jones, 2003). Writing the history before time t as $\mathcal{H}_t = \{(t_i, k_i) : t_i < t\}$, the standard object is the mark-specific conditional intensity $\lambda_k(t \mid \mathcal{H}_t)$, whose value can be interpreted as the instantaneous rate at which an event of type k occurs at time t given the past. Summing over marks gives the total intensity, $\lambda(t \mid \mathcal{H}_t) = \sum_{k=1}^K \lambda_k(t \mid \mathcal{H}_t)$, and the corresponding sequence density on an interval $[0, T]$ can be written as

$$p(\{(t_i, k_i)\}_{i=1}^n) = \left[\prod_{i=1}^n \lambda_{k_i}(t_i \mid \mathcal{H}_{t_i})\right] \exp\left(-\int_0^T \lambda(s \mid \mathcal{H}_s)\, ds\right). \tag{1}$$

This formalism makes clear that prediction has two coupled parts: forecasting *when* the next event happens and *which mark* it carries. Classical Hawkes models specify these intensities through hand-designed self- and cross-excitation kernels (Hawkes, 1971), whereas recent neural approaches learn flexible history representations from data. In our setting, the inference problem is in-context next-event or long-horizon forecasting: given a small set of reference sequences from one system and a partial history from a test sequence, the algorithm predicts the next N event times and marks without per-dataset retraining.

A.2. Datasets

For evolution, we use 33 synthetic point-process datasets generated from the same broad synthetic distribution used in FIM-PP (Berghaus et al., 2026). In particular, we reuse their point-process family and synthetic data-generation setup, which spans Poisson, Hawkes, and periodic regimes with randomized time-dependent base intensities, mark-to-mark interaction kernels, and excitatory, inhibitory, or neutral couplings between marks; our subsampled training collection contains between 1 and 22 marks and 25 sampled sequences per dataset. These synthetic tasks are meant to expose the search to qualitatively different dynamics, ranging from nearly memoryless arrivals to strongly history-dependent excitation patterns. The full `EVIL` variant additionally incorporates small subsamples from five real datasets, while `EVIL (synthetic prior)` uses only the synthetic collection.

TAXI. This dataset is derived from New York City taxi trip logs, using the benchmark preprocessing adopted in prior TPP work (Zeng et al., 2024). Each sequence corresponds to one driver's activity, and each event records either a pick-up or a drop-off together with the borough in which it occurred, yielding 10 marks in total. We follow the standard benchmark subset of 2,000 drivers.

TAOBAO. This dataset comes from user interaction logs on the Taobao e-commerce platform (Zhu et al., 2018). Each sequence tracks one anonymous user's actions over time, and the 17 marks correspond to different interaction or product-category groups. We use the benchmark subset of 2,000 active users.

STACKOVERFLOW. Here each sequence records the times at which a user receives badges on StackOverflow (Leskovec & Krevl, 2014). The marks are the 22 badge types, and the benchmark subset contains 2,200 active users.

AMAZON. This dataset consists of user review histories on Amazon over a multi-year period (Ni et al., 2019). Each event is a review timestamp, and the mark is the corresponding product category; the benchmark version uses 16 marks and 5,200 active users.

RETWEET. This dataset represents retweet activity over time (Zhou et al., 2013). Each sequence is a user's retweet history, and the three marks coarse-grain the influence of the original poster into small, medium, and large accounts. We use the benchmark subset of 5,200 active users.

MIMIC-II. For out-of-distribution evaluation, we additionally report transfer to MIMIC-II, a medical event dataset derived from intensive-care records (Lee et al., 2011). In our benchmark it contains 75 marks, far beyond the maximum of 22 marks seen during evolution, so it provides a clean test of whether the evolved heuristic scales to much richer event vocabularies without retraining.

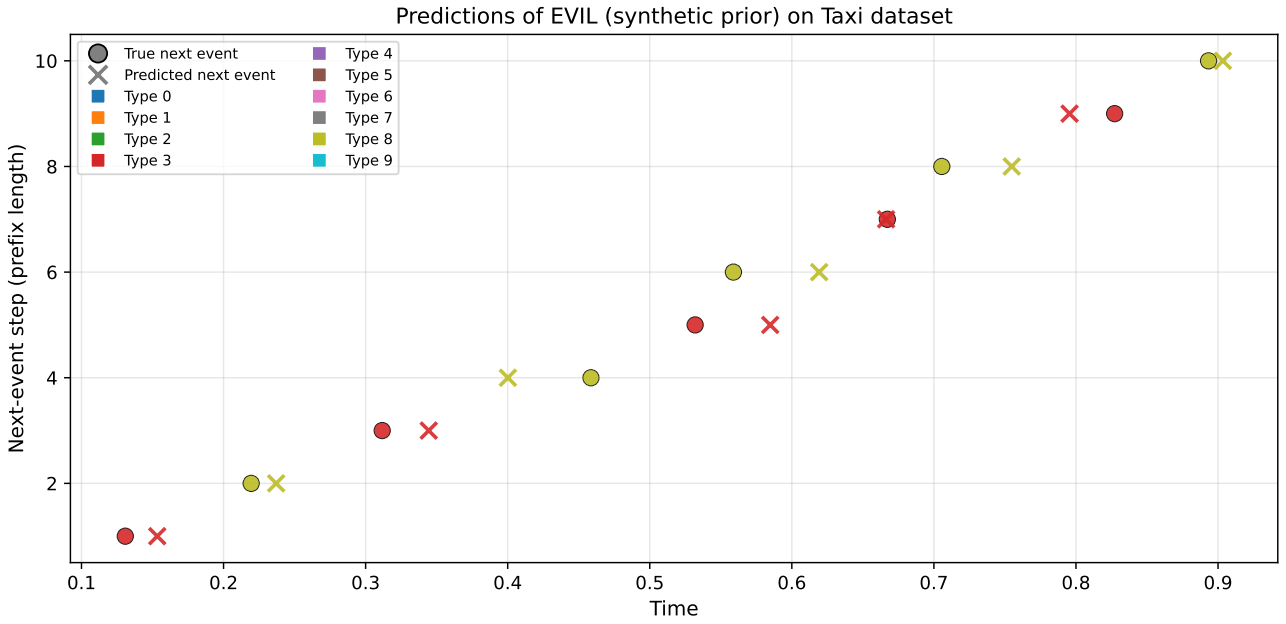

Figure 4. Illustration of mark alternation in TAXI and how the evolved next-event heuristic handles it. The dataset often alternates between marks (e.g., pick-up vs. drop-off); this pattern is not covered by FIM-PP's synthetic training data, whereas `EVIL (synthetic prior)` generalizes to it from synthetic evolution.

For all five real-world benchmark datasets, we follow the preprocessing and train/test splits used by prior TPP work, especially CDiff and FIM-PP (Zeng et al., 2024; Berghaus et al., 2026).

A.3. Evaluation Metrics

Following the benchmark protocol of CDiff and FIM-PP (Zeng et al., 2024; Berghaus et al., 2026), we evaluate predicted future sequences $\hat{\mathbf{s}}_{\text{future}} = \{(\hat{t}_i, \hat{k}_i)\}_{i=1}^{N}$ against ground truth $\mathbf{s}_{\text{future}} = \{(t_i, k_i)\}_{i=1}^{N}$. For next-event prediction, we report mark accuracy together with an error metric on the predicted inter-event time. For long-horizon forecasting, we report Optimal Transport Distance (OTD) (Mei et al., 2019), which measures the overall discrepancy between predicted and true event sequences through edit-style costs on timing, marks, and event counts.

Let $\Delta t_i = t_i - t_{i-1}$ and let $C_{j,k}$ and $\hat{C}_{j,k}$ denote the true and predicted counts of mark k in test sequence j. Then the main quantitative metrics are

$$\text{Acc} = \frac{1}{m}\sum_{j=1}^{m}\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(k_{j,i} = \hat{k}_{j,i}), \tag{2}$$

$$\text{RMSE}_e = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\sum_{k\in\mathcal{K}}(C_{j,k} - \hat{C}_{j,k})^2}, \tag{3}$$

$$\text{RMSE}_{\Delta t} = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\frac{1}{N}\sum_{i=1}^{N}(\Delta t_{j,i} - \widehat{\Delta t}_{j,i})^2}, \tag{4}$$

and

$$\text{sMAPE}_{\Delta t} = \frac{100}{m}\sum_{j=1}^{m}\frac{1}{N}\sum_{i=1}^{N}\frac{2|\Delta t_{j,i} - \widehat{\Delta t}_{j,i}|}{|\Delta t_{j,i}| + |\widehat{\Delta t}_{j,i}|}. \tag{5}$$

Here, accuracy is higher-is-better, while OTD, RMSE_e, $\text{RMSE}_{\Delta t}$, and $\text{sMAPE}_{\Delta t}$ are lower-is-better. Intuitively, Acc checks whether the predicted marks are correct at the right positions, RMSE_e compares the overall mark histogram over the forecast horizon, and the last two metrics measure the quality of the predicted inter-event times.

A.4. Baseline Models

NHP and A-NHP (Mei & Eisner, 2017; Mei et al., 2022) are neural intensity models, IFTPP (Shchur et al., 2020) is an intensity-free neural alternative, HawkesEM (Lewis & Mohler, 2011) is a classical expectation maximization baseline implemented via `tick` (Bacry et al., 2017), Dual-TPP and HYPRO (Deshpande et al., 2021; Xue et al., 2022) are long-horizon forecasting architectures, and TCDDM together with CDiff (Lin et al., 2022; Zeng et al., 2024) are generative diffusion-style approaches for event-sequence prediction. We also compare to FIM-PP (Berghaus et al., 2026), a pretrained transformer-based foundation inference model that is trained on a large synthetic corpus of point processes and then applied either zero-shot or after lightweight finetuning to a target dataset. All of these baselines are substantially more complex than our final heuristic, and except for the pretrained FIM-PP variants they require separate training for each dataset.

B. Long-Horizon Temporal Point Process Tables

Table 9. Prediction of $N = 20$ events in test sequences of five real-world datasets. Error-bars indicate the standard deviation over 10 trials. Results for the baseline methods were extracted from (Zeng et al., 2024) and (Berghaus et al., 2026). Best results are bold.

Dataset	Method	OTD	RMSE_e	$\text{RMSE}_{\Delta t}$	$\text{sMAPE}_{\Delta t}$
TAXI	HYPRO	21.653±0.163	1.231±0.015	0.372±0.004	93.803±0.454
	Dual-TPP	24.483±0.383	1.353±0.037	0.402±0.006	95.211±0.187
	A-NHP	24.762±0.217	1.276±0.015	0.430±0.003	97.388±0.381
	NHP	25.114±0.268	1.297±0.019	0.399±0.040	96.459±0.521
	IFTPP	24.053±0.609	1.364±0.032	0.384±0.005	95.719±0.779
	TCDDM	22.148±0.529	1.309±0.030	0.382±0.019	90.596±0.574
	CDiff	21.013±0.158	1.131±0.017	0.351±0.004	87.993±0.178
	FIM-PP (zs)	23.145 ±0.073	1.421 ±0.014	0.277 ±0.000	76.765 ±0.386
	FIM-PP (f)	17.914 ±0.117	**0.705** ±0.006	0.314 ±0.004	76.828 ±0.549
	EVIL (synthetic prior)	20.007 ±0.000	0.853 ±0.000	0.272 ±0.000	**70.750** ±0.000
	EVIL	**16.969** ±0.000	0.725 ±0.000	**0.264** ±0.000	72.324 ±0.000
TAOBAO	HYPRO	**44.336**±0.127	2.710±0.021	0.594±0.030	134.922±0.473
	Dual-TPP	47.324±0.541	3.237±0.049	0.871±0.005	141.687±0.431
	A-NHP	45.555±0.345	2.737±0.021	0.708±0.010	134.582±0.920
	NHP	48.131±0.297	3.355±0.030	0.837±0.009	137.644±0.764
	IFTPP	45.757±0.287	3.193±0.043	0.575±0.012	127.436±0.606
	TCDDM	45.563±0.889	2.850±0.058	0.569±0.015	126.512±0.491
	CDiff	44.621±0.139	2.653±0.022	**0.551**±0.002	125.685±0.151
	FIM-PP (zs)	64.281 ±0.077	3.949 ±0.010	1.988 ±0.006	169.687 ±0.089
	FIM-PP (f)	60.106 ±0.464	**2.428** ±0.005	16.068 ±0.109	152.528 ±0.377
	EVIL (synthetic prior)	49.979 ±0.000	3.227 ±0.000	1.558 ±0.000	**119.260** ±0.000
	EVIL	52.356 ±0.000	2.979 ±0.000	1.688 ±0.000	170.464 ±0.000
STACKOVERFLOW	HYPRO	42.359±0.170	1.140±0.014	1.554±0.010	110.988±0.559
	Dual-TPP	41.752±0.200	**1.134**±0.019	1.514±0.017	117.582±0.420
	A-NHP	42.591±0.408	1.142±0.011	1.340±0.006	108.542±0.531
	NHP	43.791±0.147	1.244±0.030	1.487±0.004	116.952±0.404
	IFTPP	46.280±0.892	1.447±0.057	1.669±0.005	115.122±0.627
	TCDDM	42.128±0.591	1.467±0.014	1.315±0.004	107.659±0.934
	CDiff	41.245±1.400	1.141±0.007	1.199±0.006	106.175±0.340
	FIM-PP (zs)	49.259 ±0.056	2.393 ±0.015	1.068 ±0.002	96.364 ±0.048
	FIM-PP (f)	**39.792** ±0.042	1.336 ±0.030	1.018 ±0.003	88.248 ±0.189
	EVIL (synthetic prior)	49.396 ±0.000	2.858 ±0.000	1.010 ±0.000	88.734 ±0.000
	EVIL	46.141 ±0.000	2.839 ±0.000	**0.977** ±0.000	**87.773** ±0.000
AMAZON	HYPRO	38.613±0.536	**2.007**±0.054	0.477±0.010	82.506±0.840
	Dual-TPP	42.646±0.752	2.562±0.202	0.482±0.012	86.453±2.044
	A-NHP	39.480±0.326	2.166±0.026	0.476±0.033	84.323±1.815
	NHP	42.571±0.293	2.561±0.060	0.519±0.023	92.053±1.553
	IFTPP	43.820±0.232	3.050±0.286	0.481±0.145	90.910±1.611
	TCDDM	42.245±0.174	2.998±0.115	0.476±0.111	83.826±1.508
	CDiff	37.728±0.199	2.091±0.163	0.464±0.086	81.987±1.905
	FIM-PP (zs)	46.219 ±0.108	2.073 ±0.012	0.464 ±0.001	128.635 ±0.398
	FIM-PP (f)	**37.208** ±0.098	**2.030** ±0.019	0.366 ±0.001	81.188 ±0.142
	EVIL (synthetic prior)	46.423 ±0.000	3.738 ±0.000	0.358 ±0.000	**61.841** ±0.000
	EVIL	43.932 ±0.000	3.277 ±0.000	**0.326** ±0.000	78.108 ±0.000
RETWEET	HYPRO	61.031±0.092	2.623±0.036	30.100±0.413	106.110±1.505
	Dual-TPP	61.095±0.101	2.679±0.026	28.914±0.300	106.900±1.293
	A-NHP	60.634±0.097	2.561±0.054	28.812±0.272	107.234±1.293
	NHP	60.953±0.079	2.651±0.045	27.130±0.224	107.075±1.398
	IFTPP	61.715±0.152	2.776±0.043	27.582±0.191	106.711±1.615
	TCDDM	60.501±0.087	2.387±0.050	27.303±0.152	106.048±0.610
	CDiff	60.661±0.101	**2.293**±0.034	27.101±0.113	106.184±1.121
	FIM-PP (zs)	60.238 ±0.161	4.172 ±0.064	24.057 ±0.050	99.069 ±0.390
	FIM-PP (f)	59.437 ±0.082	2.703 ±0.012	21.985 ±0.014	87.585 ±0.171
	EVIL (synthetic prior)	58.449 ±0.000	7.277 ±0.000	24.424 ±0.000	98.867 ±0.000
	EVIL	**57.966** ±0.000	6.102 ±0.000	**21.718** ±0.000	**85.880** ±0.000

Table 10. Prediction of $N=10$ events on five real-world datasets. Baselines from (Zeng et al., 2024; Berghaus et al., 2026). Best in bold.

Dataset	Method	OTD	RMSE_e	$\text{RMSE}_{\Delta t}$	$\text{sMAPE}_{\Delta t}$
TAXI	HYPRO	11.875±0.172	0.764±0.008	0.363±0.002	89.524±0.552
	Dual-TPP	13.058±0.220	0.966±0.011	0.395±0.003	90.812±0.497
	A-NHP	12.542±0.336	0.823±0.007	0.376±0.003	92.812±0.129
	NHP	13.377±0.184	0.922±0.009	0.397±0.005	92.182±0.384
	IFTPP	12.765±0.106	1.004±0.013	0.383±0.015	93.120±0.526
	TCDDM	11.885±0.149	1.121±0.072	0.385±0.009	90.703±0.356
	CDiff	11.004±0.191	0.785±0.007	0.350±0.002	90.721±0.291
	FIM-PP (zs)	13.820 ±0.124	1.190 ±0.013	0.281 ±0.001	78.141 ±0.414
	FIM-PP (f)	8.336 ±0.071	**0.451** ±0.006	0.291 ±0.004	75.366 ±0.160
	EVIL (synthetic prior)	9.753 ±0.000	0.585 ±0.000	0.273 ±0.000	**72.446** ±0.000
	EVIL	**8.324** ±0.000	0.453 ±0.000	**0.266** ±0.000	72.918 ±0.000
TAOBAO	HYPRO	21.547±0.138	1.527±0.035	0.591±0.019	133.147±0.341
	Dual-TPP	23.691±0.203	2.674±0.032	0.873±0.010	139.271±0.348
	A-NHP	21.683±0.215	1.514±0.015	0.608±0.011	135.271±0.395
	NHP	24.068±0.331	2.769±0.033	0.855±0.013	137.693±0.225
	IFTPP	23.195±0.039	2.429±0.045	0.602±0.037	127.411±0.573
	TCDDM	**21.012**±0.520	2.598±0.047	0.610±0.022	132.711±0.774
	CDiff	21.221±0.176	1.416±0.024	**0.535**±0.016	126.824±0.366
	FIM-PP (zs)	31.880 ±0.040	2.024 ±0.004	1.955 ±0.011	170.278 ±0.029
	FIM-PP (f)	27.974 ±0.162	**1.325** ±0.010	14.954 ±0.253	145.821 ±1.120
	EVIL (synthetic prior)	22.056 ±0.000	1.498 ±0.000	1.244 ±0.000	**121.642** ±0.000
	EVIL	23.928 ±0.000	1.518 ±0.000	1.414 ±0.000	168.930 ±0.000
STACKOVERFLOW	HYPRO	21.062±0.372	0.921±0.019	1.235±0.006	107.566±0.218
	Dual-TPP	21.229±0.394	0.936±0.013	1.223±0.010	107.274±0.200
	A-NHP	22.019±0.220	0.978±0.023	1.225±0.007	100.137±0.167
	NHP	21.655±0.314	0.970±0.014	1.266±0.003	108.867±0.361
	IFTPP	22.339±0.322	0.970±0.011	1.251±0.005	105.674±0.337
	TCDDM	22.042±0.193	1.205±0.014	1.228±0.010	108.111±0.112
	CDiff	20.191±0.455	0.916±0.010	1.180±0.003	102.367±0.267
	FIM-PP (zs)	23.527 ±0.033	1.188 ±0.005	1.039 ±0.003	92.919 ±0.556
	FIM-PP (f)	**19.938** ±0.093	**0.823** ±0.010	1.012 ±0.004	87.503 ±0.402
	EVIL (synthetic prior)	24.075 ±0.000	1.463 ±0.000	0.991 ±0.000	87.078 ±0.000
	EVIL	23.075 ±0.000	1.445 ±0.000	**0.960** ±0.000	**85.819** ±0.000
AMAZON	HYPRO	24.956±0.663	1.765±0.039	0.442±0.015	83.401±1.033
	Dual-TPP	25.929±0.280	2.098±0.101	0.475±0.008	82.352±1.285
	A-NHP	24.116±0.807	1.741±0.039	0.454±0.014	84.323±1.815
	NHP	25.730±0.497	1.843±0.053	0.491±0.048	89.135±1.092
	IFTPP	26.632±0.519	1.955±0.112	0.464±0.066	89.305±1.288
	TCDDM	25.091±0.227	1.778±0.090	0.448±0.082	82.105±1.564
	CDiff	24.230±0.287	1.766±0.079	0.450±0.049	82.124±2.094
	FIM-PP (zs)	21.736 ±0.115	1.141 ±0.010	0.449 ±0.002	120.894 ±0.393
	FIM-PP (f)	**18.428** ±0.124	**1.091** ±0.016	0.361 ±0.001	87.264 ±0.323
	EVIL (synthetic prior)	22.840 ±0.000	1.885 ±0.000	0.324 ±0.000	**59.647** ±0.000
	EVIL	21.947 ±0.000	1.701 ±0.000	0.314 ±0.000	74.389 ±0.000
RETWEET	HYPRO	31.743±0.068	1.927±0.027	33.683±0.245	105.073±0.958
	Dual-TPP	31.652±0.075	1.963±0.038	28.104±0.486	106.721±0.774
	A-NHP	30.337±0.065	1.823±0.031	26.310±0.333	106.021±1.011
	NHP	30.817±0.090	1.713±0.024	27.010±0.429	107.053±1.390
	IFTPP	31.974±0.032	1.942±0.062	28.825±0.221	106.014±0.633
	TCDDM	32.006±0.074	1.789±0.094	29.124±0.405	106.738±0.791
	CDiff	31.237±0.078	1.745±0.036	26.429±0.201	105.767±0.771
	FIM-PP (zs)	31.027 ±0.031	2.355 ±0.032	27.085 ±0.002	97.590 ±0.152
	FIM-PP (f)	30.592 ±0.037	**1.611** ±0.031	25.021 ±0.034	86.875 ±0.108
	EVIL (synthetic prior)	30.539 ±0.000	3.769 ±0.000	27.525 ±0.000	98.648 ±0.000
	EVIL	**30.330** ±0.000	3.116 ±0.000	**24.819** ±0.000	**84.946** ±0.000

Table 11. Prediction of $N=5$ events on five real-world datasets (full results). Baselines from (Zeng et al., 2024; Berghaus et al., 2026). Best in bold.

Dataset	Method	OTD	RMSE_e	$\text{RMSE}_{\Delta t}$	$\text{sMAPE}_{\Delta t}$
TAXI	HYPRO	5.952±0.126	0.500±0.011	0.322±0.004	85.994±0.227
	Dual-TPP	7.534±0.111	0.636±0.009	0.340±0.003	89.727±0.320
	A-NHP	6.441±0.090	0.682±0.010	0.347±0.002	89.070±0.152
	NHP	7.405±0.122	0.641±0.013	0.351±0.008	91.625±0.177
	IFTPP	7.209±0.184	0.608±0.008	0.335±0.003	90.512±0.169
	TCDDM	5.877±0.095	0.648±0.015	0.327±0.005	88.051±0.240
	CDiff	5.966±0.083	0.547±0.007	0.318±0.003	89.535±0.294
	FIM-PP (zs)	6.773 ±0.064	0.655 ±0.013	0.246 ±0.001	74.912 ±0.793
	FIM-PP (f)	4.083 ±0.032	0.311 ±0.007	0.250 ±0.002	71.108 ±0.902
	EVIL (synthetic prior)	4.385 ±0.000	0.343 ±0.000	**0.236** ±0.000	**69.672** ±0.000
	EVIL	**4.054** ±0.000	**0.299** ±0.000	0.241 ±0.000	71.038 ±0.000
TAOBAO	HYPRO	11.317±0.111	0.817±0.037	0.573±0.011	133.837±0.524
	Dual-TPP	13.280±0.092	1.877±0.014	0.691±0.007	134.437±0.458
	A-NHP	11.223±0.145	0.873±0.023	0.550±0.014	132.266±0.532
	NHP	11.973±0.176	1.910±0.031	0.712±0.017	134.693±0.369
	IFTPP	11.052±0.108	1.941±0.049	0.601±0.017	126.320±0.591
	TCDDM	11.609±0.184	1.690±0.023	0.675±0.009	129.009±0.923
	CDiff	10.147±0.140	**0.730**±0.019	**0.519**±0.008	124.339±0.322
	FIM-PP (zs)	15.951 ±0.042	1.129 ±0.007	1.761 ±0.013	168.299 ±0.249
	FIM-PP (f)	13.173 ±0.261	0.745 ±0.010	14.892 ±0.370	146.921 ±0.858
	EVIL (synthetic prior)	**9.894** ±0.000	**0.730** ±0.000	0.974 ±0.000	**121.309** ±0.000
	EVIL	10.938 ±0.000	0.745 ±0.000	1.167 ±0.000	166.051 ±0.000
STACKOVERFLOW	HYPRO	11.590±0.186	0.586±0.019	1.227±0.018	109.014±0.422
	Dual-TPP	11.719±0.109	0.591±0.026	1.296±0.010	106.697±0.381
	A-NHP	11.595±0.197	0.575±0.009	1.188±0.014	105.799±0.516
	NHP	11.807±0.155	0.596±0.015	1.261±0.013	108.074±0.661
	IFTPP	13.124±0.174	0.702±0.008	1.182±0.039	108.409±0.692
	TCDDM	11.410±0.129	0.630±0.015	1.201±0.028	107.893±0.942
	CDiff	10.735±0.183	0.571±0.012	1.153±0.011	100.586±0.299
	FIM-PP (zs)	11.520 ±0.057	0.657 ±0.003	1.030 ±0.001	93.296 ±0.506
	FIM-PP (f)	**10.353** ±0.051	**0.527** ±0.004	0.990 ±0.003	86.443 ±0.128
	EVIL (synthetic prior)	11.913 ±0.000	0.773 ±0.000	0.972 ±0.000	86.366 ±0.000
	EVIL	11.645 ±0.000	0.772 ±0.000	**0.946** ±0.000	**84.368** ±0.000
AMAZON	HYPRO	9.552±0.172	1.397±0.033	0.433±0.008	82.847±0.748
	Dual-TPP	11.309±0.093	1.742±0.302	0.476±0.010	86.633±0.573
	A-NHP	**9.430**±0.131	1.117±0.049	0.427±0.033	83.121±0.415
	NHP	11.273±0.198	1.431±0.024	0.501±0.009	90.591±0.667
	IFTPP	10.230±0.224	1.663±0.168	0.447±0.015	88.900±0.610
	TCDDM	10.557±0.331	1.409±0.203	0.460±0.032	82.401±0.810
	CDiff	9.478±0.081	1.326±0.082	0.424±0.018	81.287±0.994
	FIM-PP (zs)	11.124 ±0.059	**0.736** ±0.004	0.449 ±0.004	119.129 ±0.746
	FIM-PP (f)	10.034 ±0.060	0.737 ±0.006	0.341 ±0.004	78.738 ±0.339
	EVIL (synthetic prior)	11.459 ±0.000	0.982 ±0.000	**0.289** ±0.000	**57.904** ±0.000
	EVIL	10.930 ±0.000	0.894 ±0.000	0.297 ±0.000	71.182 ±0.000
RETWEET	HYPRO	16.145±0.096	1.105±0.026	27.236±0.259	103.052±1.206
	Dual-TPP	16.050±0.085	1.077±0.027	31.493±0.162	101.322±1.127
	A-NHP	16.124±0.089	1.058±0.029	29.247±0.145	105.930±1.380
	NHP	15.945±0.094	1.113±0.040	32.367±0.104	107.022±1.077
	IFTPP	16.043±0.222	1.313±0.011	30.853±0.119	106.941±2.031
	TCDDM	15.874±0.053	1.194±0.021	28.530±0.110	105.570±0.940
	CDiff	15.858±0.080	**1.023**±0.036	26.078±0.175	106.620±1.008
	FIM-PP (zs)	15.747 ±0.032	1.342 ±0.027	28.138 ±0.068	98.668 ±0.794
	FIM-PP (f)	15.645 ±0.020	1.033 ±0.034	25.308 ±0.135	83.010 ±0.278
	EVIL (synthetic prior)	15.577 ±0.000	1.908 ±0.000	27.093 ±0.000	92.250 ±0.000
	EVIL	**15.557** ±0.000	1.603 ±0.000	**24.738** ±0.000	**82.464** ±0.000

C. Ablations

C.1. Dataset-specific training

As an ablation study, we also evaluated what happens when evolution is performed on only a single dataset. The goal is to test whether the model can discover an algorithm that performs better on that specific dataset and to what extent such an algorithm still generalizes to the others. Table 12 shows the results for evolving only on Amazon. We find that on Amazon this dataset-specific variant improves slightly in $\text{sMAPE}_{\Delta t}$, while its OTD is slightly worse. At the same time, the evolved algorithm still generalizes reasonably well to the other datasets. Overall, this suggests that, at least in our setting where the model does not have direct access to the data through plots or other auxiliary views, evolving on only a single dataset does not lead to significant differences.

C.2. Variance between independent runs

To assess how sensitive the evolutionary search is to the random seed, we repeated the same setup across five independent runs with different seeds. Table 13 reports the mean and standard deviation across these runs. We observe moderate variability across datasets and metrics, which is expected for an evolutionary search procedure. At the same time, the mean results remain broadly competitive with the baselines, and the standard deviations are generally small relative to the gaps between methods. This suggests that the method is not entirely seed-invariant, but its overall performance profile is fairly stable.

C.3. Performance of a long run

We also ran a substantially longer evolution with 800 iterations instead of 100 to test whether extended search would discover qualitatively new algorithms. The resulting test-set performance is summarized in Table 14. In this longer run, the search produced 21 new best algorithms on the validation set, as shown in Figure 5, and the final discovered program was much more involved, with roughly 400 lines of code. This indicates that the search continues to explore new parts of the program space over time. However, this added complexity only resulted in tiny improvements on the validation set and no noticeable improvements on the test set. This suggests that, at least in our current setup, the search is able to find most of the performance gains within the first 100 iterations, and further search does not lead to significant improvements on the benchmark datasets. We hypothesize that this could be due to a combination of factors, including the limited size of the validation set, the inherent noise in the evaluation process, and the fact that the benchmark datasets may not be sufficiently challenging to differentiate between closely related algorithms. Additionally, we set the time limit to 5 minutes per run which means that some potentially very advanced well performing algorithms might have been discarded.

D. Additional Markov Jump Process Details

D.1. Continuous-Time Markov Jump Processes

An MJP is a continuous-time Markov chain with right-continuous, piecewise-constant trajectories on a finite state space (Norris, 1998). It is parameterized by an initial distribution π_0 and a generator (rate matrix) $Q \in \mathbb{R}^{K\times K}$ satisfying

$$Q_{ij} \geq 0 \text{ for } i \neq j, \qquad Q_{ii} = -\sum_{j\neq i} Q_{ij}. \tag{6}$$

The off-diagonal entry Q_{ij} is the instantaneous jump rate from state i to state j, while the diagonal term sets the total exit rate from state i. If we write the state-occupation probabilities as a row vector $p(t)$, then the dynamics obey the master equation

$$\frac{d}{dt}p(t) = p(t)Q, \qquad p(t) = \pi_0 \exp(Qt). \tag{7}$$

This is the main object behind our MJP experiments: once Q and π_0 are known, one can recover time-dependent state probabilities, simulate new trajectories, and compute several physically meaningful observables.

The inverse problem considered in the main text is difficult because we do not observe a full jump trajectory. Instead, we only see a discrete-time snapshot of the latent process, often on an irregular grid and possibly with observation noise. Between two consecutive observation times, the hidden process may remain in the same state or may undergo multiple unobserved jumps. Estimating Q from such partial information is therefore more challenging than estimating a discrete-time transition matrix from fully observed state sequences.

Table 12. Long-horizon prediction (N=5) on five real-world datasets. `EVIL` (Amazon-only) has only been evolved on the amazon dataset. We show OTD and sMAPE$_{\Delta t}$ (both lower is better) for the best neural baseline (CDiff), FIM-PP variants, and `EVIL` variants. Baseline results from (Zeng et al., 2024; Berghaus et al., 2026). FIM(zs) is FIM-PP (Berghaus et al., 2026) in zero-shot mode, FIM(f) is FIM-PP after finetuning on the target dataset. EVIL(s) is `EVIL (synthetic prior)`.

	TAXI		TAOBAO		STACKOV.		AMAZON		RETWEET	
Method	OTD	sM	OTD	sM	OTD	sM	OTD	sM	OTD	sM
CDiff	5.97	89.5	10.15	124.3	10.74	100.6	**9.48**	81.3	15.86	106.6
FIM(zs)	6.77	74.9	15.95	168.3	11.52	93.3	11.12	119.1	15.75	98.7
FIM(f)	4.08	71.1	13.17	146.9	**10.35**	86.4	10.03	78.7	15.65	83.0
EVIL(s)	4.39	**69.7**	**9.89**	**121.3**	11.91	86.4	11.46	**57.9**	15.58	92.3
EVIL	**4.05**	71.0	10.94	166.1	11.65	**84.4**	10.93	71.2	15.56	**82.5**
EVIL(amazon only)	**3.93**	72.5	10.24	**119.8**	12.07	86.3	11.16	**54.7**	15.57	98.7

Table 13. Long-horizon prediction (N=5) on five real-world datasets: mean ± std over five independent `EVIL` runs with different seeds. We show OTD and sMAPE$_{\Delta t}$ (both lower is better) together with the strongest neural baseline (CDiff) and FIM-PP variants. Baseline results from (Zeng et al., 2024; Berghaus et al., 2026). FIM(zs) is FIM-PP (Berghaus et al., 2026) in zero-shot mode, and FIM(f) is FIM-PP after finetuning on the target dataset.

	TAXI		TAOBAO		STACKOV.		AMAZON		RETWEET	
Method	OTD	sM	OTD	sM	OTD	sM	OTD	sM	OTD	sM
CDiff	5.97	89.5	10.15	124.3	10.74	100.6	**9.48**	81.3	15.86	106.6
FIM(zs)	6.77	74.9	15.95	168.3	11.52	93.3	11.12	119.1	15.75	98.7
FIM(f)	4.08	71.1	**10.03**	146.9	**10.35**	86.4	10.03	78.7	15.65	83.0
EVIL	4.08±0.13	71.7±3.3	10.28±0.33	131.8±17.7	11.87±0.15	**85.5**±1.3	11.36±0.25	**62.0**±6.3	**15.57**±0.07	**89.5**±6.2

Several standard quantities used in the MJP literature follow directly from the inferred generator. A stationary distribution $\pi^\star$ satisfies

$$\pi^\star Q = 0, \qquad \sum_{i=1}^{K} \pi_i^\star = 1. \tag{8}$$

The dominant relaxation times are determined by the non-zero eigenvalues $\lambda_2, \ldots, \lambda_K$ of Q through $-1/\mathrm{Re}(\lambda_i)$, which characterize how quickly perturbations decay toward stationarity. Mean first-passage times can likewise be obtained by solving linear systems associated with the generator. In addition, exact trajectories can be sampled with the Gillespie stochastic simulation algorithm, which alternates between sampling an exponential waiting time and a discrete next state according to the outgoing rates (Gillespie, 1977).

D.2. Synthetic Training Data Reused from FIM-MJP

For the MJP task, we reuse the synthetic training corpus introduced in FIM-MJP (Berghaus et al., 2024) rather than generating a new synthetic benchmark from scratch. Their data-generation pipeline samples MJPs on state spaces of size 2–6 by first drawing a connected adjacency structure and then sampling the allowed off-diagonal rates from a small family of Beta priors. The initial distribution is chosen either as the stationary distribution of the sampled generator or from a Dirichlet distribution, so the resulting processes cover both equilibrium-like and more generic initializations.

Trajectories are then simulated with the Gillespie algorithm on a time horizon of length 10 (Gillespie, 1977). To obtain observations, the latent paths are recorded on a base grid with at most 100 time points and then subsampled to produce both regular and irregular observation patterns. Finally, a small amount of label noise is injected by randomly replacing a fraction of observed states; in the version we use, this corruption level is 1%. The full FIM-MJP corpus contains 45K synthetic processes with 300 sampled paths per process (Berghaus et al., 2024).

Table 14. Long-horizon prediction (N=5) on five real-world datasets for a longer `EVIL` run with 800 iterations. We show OTD and $\text{sMAPE}_{\Delta t}$ (both lower is better) together with the strongest neural baseline (CDiff), FIM-PP variants, and the standard `EVIL` variants. Baseline results from (Zeng et al., 2024; Berghaus et al., 2026). FIM(zs) is FIM-PP (Berghaus et al., 2026) in zero-shot mode, FIM(f) is FIM-PP after finetuning on the target dataset, and EVIL(s) is `EVIL (synthetic prior)`.

	TAXI		TAOBAO		STACKOV.		AMAZON		RETWEET	
Method	OTD	sM	OTD	sM	OTD	sM	OTD	sM	OTD	sM
CDiff	5.97	89.5	10.15	124.3	10.74	100.6	**9.48**	81.3	15.86	106.6
FIM(zs)	6.77	74.9	15.95	168.3	11.52	93.3	11.12	119.1	15.75	98.7
FIM(f)	4.08	71.1	13.17	146.9	**10.35**	86.4	10.03	78.7	15.65	83.0
EVIL(s)	4.39	**69.7**	**9.89**	**121.3**	11.91	86.4	11.46	**57.9**	15.58	92.3
EVIL	**4.05**	71.0	10.94	166.1	11.65	**84.4**	10.93	71.2	15.56	**82.5**
EVIL (800 iterations)	4.06	71.6	9.93	125.0	11.90	86.3	11.36	67.8	**15.40**	90.8

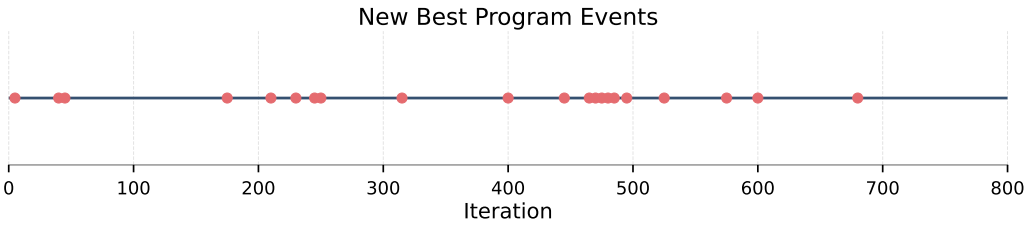

Figure 5. New best validation-set programs discovered during the 800-iteration long run. Across the run, the search found 21 new best algorithms, indicating continued progress on the validation objective even though the final gains on the benchmark test datasets remain small.

D.3. MJP Datasets

Our MJP experiments cover four qualitatively different settings. Together they test whether a single zero-shot heuristic can handle synthetic non-equilibrium physics, noisy biophysical recordings, and coarse-grained molecular dynamics.

Discrete Flashing Ratchet (DFR). The DFR is a synthetic six-state benchmark derived from a Brownian ratchet model (Ajdari & Prost, 1992). It represents a particle moving on a periodic asymmetric landscape whose potential is switched on and off. In the "on" configuration, the asymmetry induces a directional bias; in the "off" configuration, the particle diffuses more freely. The switching breaks detailed balance and creates a non-equilibrium steady state with a persistent current, making the DFR a useful benchmark for testing whether inferred generators capture genuinely irreversible dynamics. In our benchmark setup, we follow the dataset used by prior MJP work (Seifner & Sánchez, 2023; Berghaus et al., 2024): 5000 coarse-grained paths observed on irregular grids with 50 observation times per path. Section D.5 provides the mathematical details of the DFR construction and the entropy-production calculation.

Ion Channel (IonCh). This dataset comes from experimental recordings of the viral potassium channel $\text{Kcv}_{\text{MT325}}$, which exhibits switching between a small number of metastable conductance states (Gazzarrini et al., 2006). The raw signal is continuous and noisy, so the MJP is only obtained after coarse-graining the recordings into discrete channel states. This makes the task harder than the DFR benchmark: besides inferring transition rates, one must cope with measurement noise and the fact that the latent discrete representation is itself model-dependent. The benchmark introduced in the earlier MJP papers compares methods on this three-state switching system (Seifner & Sánchez, 2023; Berghaus et al., 2024).

Alanine Dipeptide (ADP). Alanine dipeptide is a canonical molecular-kinetics benchmark because its slow conformational dynamics can be summarized well by the Ramachandran torsion angles ϕ and ψ while still exhibiting non-trivial metastability (Mironov et al., 2019). The observations are continuous molecular descriptors, typically encoded through trigonometric features of these angles, and are then mapped to a small discrete state space for MJP inference. In the benchmark used by previous work, the goal is to recover the coarse-grained switching dynamics among six metastable conformations from all-atom molecular simulation data (Mardt et al., 2017; Husic et al., 2020; Seifner & Sánchez, 2023; Berghaus et al., 2024).

Protein Folding (PFold). The PFold benchmark is a simpler two-state molecular system, originating from a Brownian-dynamics model with a bistable potential that mimics transitions between folded and unfolded configurations (Mardt et al., 2017). After coarse-graining, the task reduces to recovering a two-state generator whose rates determine the switching statistics and stationary occupancy. This dataset is useful because it isolates the basic metastable-switching regime: strong methods should match the correct time scales even when the state space is very small.

For DFR, the ground-truth process is known analytically, so one can directly evaluate parameter recovery. For IonCh, ADP, and PFold, the discrete-state abstraction is obtained from real-valued or molecular trajectories, so evaluation is more indirect and relies on whether the inferred generator reproduces the observed long-time and short-time statistics, as well as the quality of simulated rollouts.

D.4. Evaluation Metrics

We use two different types of evaluation for the MJP task. On the synthetic DFR benchmark, where the underlying parameters are known, Table 5 reports the inferred parameters as ratios to the ground truth, *e.g.* $\hat{V}/V$, $\hat{r}/r$, and $\hat{b}/b$. A value of 1 therefore corresponds to perfect recovery, values above 1 indicate overestimation, and values below 1 indicate underestimation.

For the broader rollout comparison across DFR, IonCh, ADP, and PFold, we use the Hellinger distance between empirical state distributions (Hellinger, 1909), following its use as an MJP evaluation metric in FIM-MJP (Berghaus et al., 2024). Intuitively, it measures how similar the predicted path distribution is to the true path distribution over time. This metric is especially useful for the real-world datasets, where the true latent generator is not known and parameter error therefore cannot be computed directly. Given two discrete distributions $P = (p_1, \dots, p_K)$ and $Q = (q_1, \dots, q_K)$, the Hellinger distance is

$$H(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{K} (\sqrt{p_i} - \sqrt{q_i})^2}. \tag{9}$$

In our setting the distributions are not available in closed form, so we approximate them from sampled or observed paths. At each observation time τ_ℓ, we form normalized histograms h_ℓ^{tar} and h_ℓ^{mod} over the discrete states using the target process and the model-generated process, respectively. We then compute the local discrepancy $H(h_\ell^{\mathrm{tar}}, h_\ell^{\mathrm{mod}})$ and average it over the observation grid:

$$d_{\mathrm{Hell}} = \frac{1}{L} \sum_{\ell=1}^{L} H\big(h_\ell^{\mathrm{tar}}, h_\ell^{\mathrm{mod}}\big)\,. \tag{10}$$

Lower values indicate that the simulated process matches the empirical occupancy statistics more closely over time. Following FIM-MJP (Berghaus et al., 2024), we report means and standard deviations over 100 repeated histogram comparisons.

D.5. DFR and Entropy Production

The discrete flashing ratchet (DFR) is a six-state non-equilibrium MJP obtained by coarse-graining a Brownian particle moving in an asymmetric periodic potential that is switched on and off (Ajdari & Prost, 1992). As in the FIM-MJP setup (Berghaus et al., 2024), the "on" states correspond to motion in the asymmetric potential and the "off" states to freer diffusion. The transition rates are parameterized by the potential strength V, the switching rate r, and a background diffusion rate b. For the within-mode dynamics one writes

$$f_{ij}^{\mathrm{on}} = \exp\!\left(-\frac{V}{2}(j - i)\right), \qquad f_{ij}^{\mathrm{off}} = b, \tag{11}$$

for positions $i, j \in \{1, 2, 3\}$ within the on and off sectors, respectively, while the on/off switching itself occurs at rate r between corresponding positions. Writing the six states as $\{1_{\mathrm{on}}, 2_{\mathrm{on}}, 3_{\mathrm{on}}, 1_{\mathrm{off}}, 2_{\mathrm{off}}, 3_{\mathrm{off}}\}$, the exact generator is equivalently

specified by its nonzero entries:

$$Q_{i_{\text{on}},j_{\text{on}}} = \exp\left(-\frac{V}{2}(j-i)\right), \qquad i \neq j,\ i,j \in \{1,2,3\}, \tag{12}$$

$$Q_{i_{\text{off}},j_{\text{off}}} = b, \qquad i \neq j,\ i,j \in \{1,2,3\}, \tag{13}$$

$$Q_{i_{\text{on}},i_{\text{off}}} = Q_{i_{\text{off}},i_{\text{on}}} = r, \qquad i \in \{1,2,3\}, \tag{14}$$

$$Q_{\alpha,\beta} = 0, \qquad \text{otherwise}, \tag{15}$$

$$Q_{\alpha,\alpha} = -\sum_{\beta \neq \alpha} Q_{\alpha,\beta}. \tag{16}$$

Setting $(V, r, b) = (1, 1, 1)$ recovers the ground-truth matrix used in the benchmark tables up to rounding. Intuitively, increasing V increases the asymmetry of the landscape, b sets the scale of unbiased motion when the potential is off, and r controls how often the system alternates between these two regimes.

This model is useful because the asymmetry and random switching create a stationary current, so the process is out of equilibrium. In other words, the DFR does not merely have a stationary distribution; it also exhibits irreversible probability flow around the state space. This makes it a natural benchmark for testing whether an inferred generator recovers not only occupancies, but also the directional structure of the dynamics.

For finite-state MJPs, the instantaneous total entropy production rate can be written in closed form from the generator and the time-dependent probabilities (Seifert, 2012):

$$\dot{S}_{\text{tot}}(t) = \frac{1}{2}\sum_{i,j}\left[p_i(t)Q_{ij} - p_j(t)Q_{ji}\right]\log\frac{p_i(t)Q_{ij}}{p_j(t)Q_{ji}}. \tag{17}$$

The total entropy production over a time horizon $[0, T]$ is then

$$\Sigma_T = \int_0^T \dot{S}_{\text{tot}}(t)\, dt. \tag{18}$$

At equilibrium, detailed balance makes this quantity vanish. For the DFR, by contrast, the ratchet current leads to positive entropy production, which is why it is a meaningful downstream observable in our experiments. Once a method has inferred Q and π_0, it can solve the master equation for $p(t)$ and evaluate the expression above without any extra training. The entropy-production plot in the main text should therefore be read as a derived scientific quantity that tests whether the inferred MJP captures the correct irreversible dynamics.

E. Additional Imputation Dataset Details

Our imputation experiments follow the same benchmark datasets and preprocessing conventions used by the FIM imputation work (Seifner et al., 2025). We group the datasets into point-wise missingness tasks, where individual values are masked independently, and window-missing tasks, where a contiguous middle segment is removed.

E.1. Point-Wise Missing Datasets

Eight datasets use 50% random point-wise missing patterns. Following (Seifner et al., 2025), the GUANGZHOU traffic-speed dataset (214 channels, 500 observations) and the SOLAR solar-power dataset (137 channels, 52,560 observations) come from the benchmark preprocessing used by BayOTIDE (Fang et al., 2023). The remaining six datasets come from TSI-Bench (Du et al., 2024): BEIJING air quality (132 channels, 1,458 observations), ITALY air quality (13 channels, 774 observations), ELECTRICITY consumption (370 channels, 1,457 observations), ETT_H1 electricity transformer temperature / load (7 channels, 358 observations), PEMS road occupancy (862 channels, 727 observations), and PEDESTRIAN activity in Australia (1 channel, 3,633 observations). As in the benchmark protocol, the datasets are split into train, validation, and test subsets, and 50% of each subset is randomly masked and used for evaluation.

E.2. Window-Missing Datasets

For contiguous-gap imputation, we use the Motion Capture dataset of human walking motions (Wang et al., 2007), as preprocessed by (Yildiz et al., 2019), following the benchmark setup of (Heinonen et al., 2018; Seifner et al., 2025).

The dataset contains 43 human motion trajectories, each with 50 channels and 100 observations. Roughly 20% of the middle portion of each time series is removed, creating a temporally missing window that must be reconstructed from the surrounding context. We report two variants of this benchmark: *MC (PCA)*, where the sequences are projected to a lower-dimensional PCA representation before imputation, and *MC (No PCA)*, where imputation is performed directly in the original data space.

F. Discovered Algorithms

Pseudocode 1: `EVIL` heuristic for Point Processes

Input: context event sequences, target prefixes, number of marks K.
Context statistics.

1. Aggregate all context inter-event gaps and all mark transitions $a \to b$.
2. Estimate a robust global gap statistic from all observed gaps.
3. For each previous mark a, estimate the average next-event gap after observing a.
4. Build a smoothed transition table $P(b \mid a)$ from the context sequences.
5. Estimate the global majority mark as a fallback.

Prediction for one target prefix.

1. If the prefix is empty, predict the global gap and the global majority mark.
2. Otherwise, let a be the last observed mark and compute a recency-weighted average of the most recent gaps in the prefix.
3. Predict the next gap by mixing this local recent-gap estimate with the context-level average gap associated with mark a; clamp extreme values.
4. Predict the next time as the last observed time plus the predicted gap.
5. Predict the next mark by combining local counts of transitions out of a in the prefix, mark frequencies within the prefix, and the smoothed context transition row $P(\cdot \mid a)$.
6. Return the predicted next time and the highest-scoring next mark.

Algorithm 1: `EVIL` algorithm for Point Processes

```python
"""
Zero-shot MTPP next-event prediction heuristic (evolvable block).

The function predict_next_events is called by the evaluator with batched
target histories and a context pool; it must return predicted next times and marks.
"""

import numpy as np

def predict_next_events(
    target_history_times,
    target_history_marks,
    context_times,
    context_marks,
    num_classes,
):
    b = len(target_history_times)
    pt = np.zeros(b, np.float64)
```

```
    pm = np.zeros(b, np.int64)

    # Context: robust gap, majority mark, transition matrix, and per-mark mean delta
    deltas = []
    counts = np.zeros(num_classes, np.int64)
    trans = np.zeros((num_classes, num_classes), np.float64)
    g_sum = np.zeros(num_classes, np.float64)
    g_cnt = np.zeros(num_classes, np.int64)

    for ta, ma in zip(context_times, context_marks):
        t = np.asarray(ta, np.float64)
        m = np.asarray(ma, np.int64)
        if t.size >= 2:
            d = np.diff(t)
            deltas.extend(d.tolist())
            # accumulate mean delta conditioned on the previous mark
            for prev_mark, dd in zip(m[:-1], d):
                if 0 <= prev_mark < num_classes:
                    g_sum[int(prev_mark)] += float(dd)
                    g_cnt[int(prev_mark)] += 1
        if m.size:
            counts += np.bincount(m, minlength=num_classes)
        if m.size >= 2:
            a = m[:-1]
            b2 = m[1:]
            for x, y in zip(a, b2):
                if 0 <= x < num_classes and 0 <= y < num_classes:
                    trans[int(x), int(y)] += 1.0

    ggap = float(np.median(deltas)) if deltas else 1.0
    gmaj = int(np.argmax(counts)) if counts.sum() > 0 else 0

    # Prepare a smoothed global transition baseline to avoid zero rows
    trans_baseline = trans + 1.0  # Laplace-like smoothing
    trans_baseline /= trans_baseline.sum(axis=1, keepdims=True) + 1e-12

    # Per-mark global mean delta (fallback)
    g_mean_delta = np.full(num_classes, ggap, dtype=np.float64)
    mask = g_cnt > 0
    g_mean_delta[mask] = g_sum[mask] / g_cnt[mask]

    for i in range(b):
        t = np.asarray(target_history_times[i], np.float64)
        m = np.asarray(target_history_marks[i], np.int64)
        n = t.size

        if n == 0:
            pt[i] = ggap
            pm[i] = gmaj
            continue

        last_t = float(t[-1])
        last_m = int(m[-1]) if m.size else -1

        # Time prediction: recency-weighted gaps with mark-conditioned fallback and
        ↪  clamping
        if n >= 2:
            d = np.diff(t)
            recent = d[-5:]  # up to last 5 intervals
            Lr = recent.size
            # exponential recency weights (more recent -> higher)
            exps = np.exp(np.linspace(-1.0, 0.0, max(1, Lr)))
            exps = exps[-Lr:]
            exps = exps / (exps.sum() + 1e-12)
```

```
            rec_est = float(np.dot(recent, exps))
            if 0 <= last_m < num_classes and g_cnt[last_m] > 0:
                cond_mean = g_mean_delta[last_m]
                base = 0.6 * rec_est + 0.4 * cond_mean
            else:
                base = 0.75 * rec_est + 0.25 * ggap
            base = float(max(base, 1e-6))
            # cap extreme predictions to a reasonable multiple
            cap = 20.0 * max(rec_est if rec_est > 0 else ggap, ggap)
            base = min(base, cap)
            pt[i] = last_t + base
        else:
            # Single time in history -> use per-mark mean delta if available, else global median
            if 0 <= last_m < num_classes and g_cnt[last_m] > 0:
                delta = float(g_mean_delta[last_m])
            else:
                delta = ggap
            delta = max(delta, 1e-6)
            pt[i] = last_t + delta

        # Mark prediction: combine local transition counts, local frequency, and global smoothed transitions
        scores = np.zeros(num_classes, dtype=np.float64)

        # 1) local transitions conditioned on last mark
        if m.size >= 2:
            prev = m[:-1]
            nxt = m[1:]
            mask_prev = prev == last_m
            if np.any(mask_prev):
                counts_local_next = np.bincount(nxt[mask_prev], minlength=num_classes).astype(np.float64)
                scores += 1.0 * counts_local_next

        # 2) local frequency in the sequence (helps for short histories)
        hist_counts = np.bincount(m, minlength=num_classes).astype(np.float64)
        scores += 0.25 * hist_counts

        # 3) global smoothed transition probabilities from last mark (as soft prior)
        if 0 <= last_m < num_classes:
            scores += 2.0 * trans_baseline[last_m]
        else:
            # if last mark invalid, fallback to global popularity
            scores += 0.5 * (counts + 1.0)

        # If everything is zero (very unlikely), fallback to global majority
        if scores.sum() <= 0:
            pm[i] = gmaj
        else:
            pm[i] = int(np.argmax(scores))

    return pt, pm
```

Pseudocode 2: `EVIL (synthetic prior)` heuristic for Point Processes

Input: context event sequences, target prefixes, number of marks K.
Context statistics.

1. Aggregate all inter-event gaps from the context sequences.
2. Build first-order transition counts $a \to b$ between consecutive marks.

3. Build second-order transition counts $(a, b) \to c$ for consecutive mark pairs.
4. For each previous mark a, estimate a typical outgoing gap; for each next mark b, estimate a typical incoming gap.
5. For each directed edge $a \to b$, estimate an edge-specific typical gap whenever enough examples exist.
6. Estimate a global fallback gap and a global majority mark.

Prediction for one target prefix.

1. Predict the next mark using a reliability hierarchy:
   (a) if the last two marks form a previously observed pair, use the second-order transition table;
   (b) otherwise, if the last mark has outgoing transitions in the context data, use the first-order transition table;
   (c) otherwise, fall back to the global majority mark.
2. Predict the next gap using an edge-specific timing estimate for the predicted transition whenever available.
3. If no edge-specific estimate is available, combine the typical outgoing gap of the last mark with the typical incoming gap of the predicted mark.
4. Mix this context-level timing estimate with the recent local gaps from the target prefix.
5. Predict the next time as the last observed time plus the resulting gap.
6. Return the predicted next time and mark.

Algorithm 2: `EVIL (synthetic prior)` algorithm for Point Processes

```python
"""
Zero-shot MTPP next-event prediction heuristic (evolvable block).

The function predict_next_events is called by the evaluator with batched
target histories and a context pool; it must return predicted next times and marks.
"""

import numpy as np

def predict_next_events(
    target_history_times,
    target_history_marks,
    context_times,
    context_marks,
    num_classes,
):
    b = len(target_history_times)
    # Context statistics: global, 1st- and 2nd-order mark transitions, and
    ↪  mark-conditional deltas
    trans = np.zeros((num_classes, num_classes), dtype=np.int64)
    per_prev = [[] for _ in range(num_classes)]
    per_next = [[] for _ in range(num_classes)]
    pair = {}
    edge_dt = {}
    all_dt, all_m = [], []
    for t, m in zip(context_times, context_marks):
        Lm = len(m)
        if len(t) > 1:
            dt = np.diff(t)
            all_dt.extend(dt.tolist())
            L = min(len(dt), Lm - 1) if Lm > 0 else 0
```

```
            if L > 0:
                prev = m[:L]
                nxt = m[1:1 + L]
                for j in range(L):
                    d = float(dt[j])
                    a = int(prev[j])
                    bmark = int(nxt[j])
                    per_prev[a].append(d)
                    per_next[bmark].append(d)
                    trans[a, bmark] += 1
                    edge_dt.setdefault((a, bmark), []).append(d)
            if Lm > 2:
                for j in range(Lm - 2):
                    key = (int(m[j]), int(m[j + 1]))
                    if key in pair:
                        pair[key][m[j + 2]] += 1
                    else:
                        cnt = np.zeros(num_classes, dtype=np.int64)
                        cnt[m[j + 2]] = 1
                        pair[key] = cnt
        if Lm > 0:
            all_m.extend(m)
    gmd = float(np.median(all_dt)) if all_dt else 1.0
    gmm = int(np.bincount(all_m).argmax()) if all_m else 0
    pmd = np.full(num_classes, gmd, dtype=float)
    nmd = np.full(num_classes, gmd, dtype=float)
    for c in range(num_classes):
        if per_prev[c]:
            pmd[c] = float(np.median(per_prev[c]))
        if per_next[c]:
            nmd[c] = float(np.median(per_next[c]))
    next_by_prev = (trans + 1).argmax(axis=1)  # Laplace smoothing
    row_counts = trans.sum(axis=1)
    # per-transition median delta (edge-specific)
    edge_md = {k: float(np.median(v)) for k, v in edge_dt.items()} if edge_dt else {}

    T = np.zeros(b, dtype=np.float64)
    M = np.zeros(b, dtype=np.int64)
    for i in range(b):
        th = target_history_times[i]
        mh = target_history_marks[i]
        ln = len(mh)
        lm = int(mh[-1]) if ln > 0 else -1

        # Predict mark with reliability checks
        if ln >= 2:
            key = (int(mh[-2]), lm)
            if key in pair and pair[key].sum() > 0:
                nm = int(pair[key].argmax())
            elif 0 <= lm < num_classes and row_counts[lm] > 0:
                nm = int(next_by_prev[lm])
            else:
                nm = gmm
        elif ln == 1:
            if 0 <= lm < num_classes and row_counts[lm] > 0:
                nm = int(next_by_prev[lm])
            else:
                nm = gmm
        else:
            nm = gmm
        M[i] = int(nm)

        # Predict time using local recent median and edge-specific/context medians
        if len(th) > 0:
```

```
            base = float(th[-1])
            # reference delta from edge if available, else combine per-prev and per-next, else global
            if ln > 0 and 0 <= lm < num_classes and 0 <= nm < num_classes:
                ek = (lm, nm)
                if ek in edge_md:
                    dref = float(edge_md[ek])
                else:
                    dref = 0.5 * (pmd[lm] + nmd[nm])
            else:
                dref = gmd

            if len(th) > 1:
                k = min(5, len(th))
                dloc = float(np.median(np.diff(th[-k:])))
                alpha = 0.65 if len(th) >= 3 else 0.5
                d = alpha * dloc + (1.0 - alpha) * dref
            else:
                d = dref
            T[i] = base + d
        else:
            T[i] = gmd
    return T, M
```

Pseudocode 3: `EVIL (synthetic prior)` heuristic for Markov Jump Processes

Input: partially observed trajectories, observation times, sequence lengths, number of states K.
Initial distribution.

1. Count the first observed state of each trajectory.
2. Add smoothing, and optionally add a smaller contribution from second observations to reduce sensitivity to noise.
3. Normalize the counts to obtain the initial-state distribution.

Characteristic time scale.

1. Collect valid time differences between consecutive observations.
2. Use a robust typical interval, such as the median positive difference.

Rate matrix estimation.

1. For each observed interval spent in state i, add its duration to the exposure time of state i, while capping unusually long intervals.
2. Count observed exits from each state, ignoring implausibly fast changes that are likely due to noise.
3. For each valid transition $i \to j$, accumulate a destination count, giving larger weight to transitions observed over shorter intervals.
4. Estimate each state's exit hazard as a smoothed ratio of exits to exposure time, then clip it to a reasonable range.
5. Normalize destination counts from each state to obtain off-diagonal transition probabilities.
6. Form the rate matrix by multiplying each state's exit hazard with its destination distribution, then set the diagonal so that each row sums to zero.

Algorithm 3: `EVIL (synthetic prior)` algorithm for Markov Jump Processes

```python
"""
Zero-shot MJP parameter estimation heuristic (evolvable block).
"""

import numpy as np

def estimate_mjp_parameters(
    observation_grid: np.ndarray,
    observation_values: np.ndarray,
    seq_lengths: np.ndarray,
    n_states: int,
):
    """
    Predict the global rate matrix and initial distribution for one MJP sample.
    Inputs are for a single sample: num_paths trajectories, each with up to max_seq_len observations.

    Args:
        observation_grid: (num_paths, max_seq_len) array of observation times.
        observation_values: (num_paths, max_seq_len) array of observed discrete states (0 to n_states-1).
        seq_lengths: (num_paths,) array indicating the number of valid observations per path.
        n_states: The number of states for this MJP.

    Returns:
        Tuple of:
        - pred_rate_matrix: (n_states, n_states) array. Off-diagonals are >= 0.
        - pred_init_dist: (n_states,) array of probabilities summing to 1.
    """
    batch_size, max_len = observation_grid.shape

    # ------------------------------------------------------------------------
    # 1. Estimate Initial Distribution globally (Vectorized)
    # ------------------------------------------------------------------------
    valid_paths = seq_lengths > 0
    if not np.any(valid_paths):
        # Fallback if entirely empty sequence batch
        return np.zeros((n_states, n_states)), np.ones(n_states) / n_states

    # Grab first observations with uniform prior
    first_obs = observation_values[valid_paths, 0].astype(int)
    counts = np.bincount(first_obs, minlength=n_states).astype(np.float64) + 1.0

    # Grab second observations (if available) with reduced weight to smooth noise
    valid_len2 = seq_lengths > 1
    if np.any(valid_len2):
        second_obs = observation_values[valid_len2, 1].astype(int)
        counts += np.bincount(second_obs, minlength=n_states) * 0.5

    global_init_dist = counts / np.sum(counts)

    # ------------------------------------------------------------------------
    # 2. Extract Valid Intervals & Estimate Time Scale (Vectorized)
    # ------------------------------------------------------------------------
    # Create a boolean mask of valid transitions: (batch, max_len - 1)
    mask = np.arange(max_len - 1)[None, :] < (seq_lengths[:, None] - 1)

    diffs = np.diff(observation_grid, axis=1)
    valid_dts = diffs[mask]

    pos_dts = valid_dts[valid_dts > 0]
    typical_dt = float(np.median(pos_dts)) if pos_dts.size > 0 else 1.0
```

```
    typical_dt = max(1e-8, typical_dt)

    # ------------------------------------------------------------------------
    # 3. Accumulate State Exits and Exposures
    # ------------------------------------------------------------------------
    # Extract flattened arrays of valid states and times
    curr_states = observation_values[:, :-1][mask].astype(int)
    next_states = observation_values[:, 1:][mask].astype(int)
    dts = valid_dts

    # Extract the last valid state of each path for right-censoring exposure
    last_idx = np.maximum(0, seq_lengths - 1)
    last_states = observation_values[np.arange(batch_size),
    ↪  last_idx][valid_paths].astype(int)

    # Compute total exposure time per state (capped to prefer small/medium intervals)
    capped_dts = np.minimum(dts, 2.0 * typical_dt)
    time_small = np.bincount(curr_states, weights=capped_dts, minlength=n_states)
    time_small += np.bincount(last_states, weights=np.full(last_states.shape, 0.5 *
    ↪  typical_dt), minlength=n_states)

    # Filter out transitions that are too fast (likely noise)
    valid_change = (curr_states != next_states) & (dts > 0.02 * typical_dt)
    change_small = np.bincount(curr_states[valid_change], minlength=n_states)

    # Accumulate destination targets, weighted by recency
    dest_counts = np.zeros((n_states, n_states), dtype=np.float64)
    weights = np.exp(-dts[valid_change] / typical_dt)
    np.add.at(dest_counts, (curr_states[valid_change], next_states[valid_change]),
    ↪  weights)

    # ------------------------------------------------------------------------
    # 4. Assemble Rate Matrix
    # ------------------------------------------------------------------------
    # Estimate per-state exit rates (hazard)
    alpha = (change_small + 0.1) / (time_small + 0.25 * typical_dt)
    alpha = np.clip(alpha, 1e-8 / typical_dt, 10.0 / typical_dt)

    # Compute destination transition probabilities
    P_off = dest_counts + (0.05 / max(1, n_states - 1))
    np.fill_diagonal(P_off, 0.0) # Ensure no self-loops exist in the probability
    ↪  matrix

    row_sums = P_off.sum(axis=1, keepdims=True)
    row_sums[row_sums == 0] = 1.0  # Prevent division by zero if a state has no
    ↪  outward paths
    P_off /= row_sums

    # Construct the final Q-matrix: Q = diag(alpha) * P_off
    global_rate_matrix = alpha[:, None] * P_off

    # Strictly enforce Q-matrix eigenvalue constraint: rows must sum to exactly zero
    np.fill_diagonal(global_rate_matrix, 0.0)
    global_rate_matrix[np.diag_indices(n_states)] = -global_rate_matrix.sum(axis=1)

    return global_rate_matrix, global_init_dist
```

Pseudocode 4: EVIL heuristic for Time Series Imputation

Input: partially observed time series $x_{1:T}$ with timestamps.
Process each dimension independently.

1. Detect contiguous blocks of missing values.

2. If the series is entirely missing, fill it with a simple constant fallback.
3. For each missing block:
   (a) If the gap is short, leave it for the interpolation fallback.
   (b) If the gap is long enough and sufficient observed context exists before it, extract the observed window immediately preceding the gap.
   (c) Search earlier in the same series for candidate windows whose preceding context is fully observed and has the same length.
   (d) Score each candidate by how well its preceding context matches the current pre-gap context.
   (e) Copy the continuation that follows the best matching window into the gap.
   (f) Apply a level shift so the copied pattern connects smoothly to the value right before the gap.
4. After motif retrieval has been attempted for all long gaps, fill any remaining missing positions by time-aware linear interpolation.

Algorithm 4: EVIL algorithm for time series imputation

```python
"""
Time series imputation heuristic (evolvable block).

Data contains randomly masked point-wise missing patterns.
The function receives `observation_values` (T, D) | batched across D channels within one dataset |
with NaN at missing/holdout positions, `observation_times` (T,) per time step, and
`prediction_mask` (T, D) True at holdout positions we must predict.
"""

import numpy as np
from numpy.lib.stride_tricks import sliding_window_view

def _impute_1d(out: np.ndarray, times: np.ndarray) -> None:
    """
    Time-aware hybrid imputation for a single 1D series.
    Uses vectorized motif retrieval for large gaps and linear interpolation for small gaps.
    """
    large_gap_threshold = 4
    context_size = 8

    valid = np.isfinite(out)

    if not np.any(valid):
        out[:] = 0.0
        return
    if np.all(valid):
        return

    # 1. Identify contiguous missing blocks (vectorized)
    is_missing = ~valid
    padded = np.concatenate(([False], is_missing, [False]))
    starts = np.where(padded[1:] & ~padded[:-1])[0]
    ends = np.where(~padded[1:] & padded[:-1])[0]

    for s, e in zip(starts, ends):
        gap_len = e - s

        if gap_len >= large_gap_threshold and s >= context_size:
            context = out[s - context_size : s]

```

```
            if np.all(np.isfinite(context)):
                req_len = context_size + gap_len
                search_end = s - req_len + 1

                if search_end > 0:
                    # Search space is strictly before the current gap
                    search_space_valid = valid[:search_end + req_len - 1]

                    # 2. Vectorized identification of valid windows
                    windows_valid = sliding_window_view(search_space_valid,
                    ↪  req_len).all(axis=1)
                    valid_indices = np.where(windows_valid)[0]

                    if len(valid_indices) > 0:
                        # 3. Vectorized extraction of candidate contexts
                        search_space_vals = out[:search_end + context_size - 1]
                        all_cand_contexts = sliding_window_view(search_space_vals,
                        ↪  context_size)

                        valid_cand_contexts = all_cand_contexts[valid_indices]

                        # 4. Vectorized distance calculation
                        distances = np.sum((valid_cand_contexts - context)**2,
                        ↪  axis=1)

                        best_idx_in_valid = np.argmin(distances)
                        best_match_idx = valid_indices[best_idx_in_valid]

                        # 5. Inject the retrieved motif
                        retrieved_pattern = out[best_match_idx + context_size :
                        ↪  best_match_idx + context_size + gap_len]

                        # Apply level shift
                        shift = context[-1] - out[best_match_idx + context_size - 1]
                        out[s : e] = retrieved_pattern + shift

                        # Mark newly imputed block as valid
                        valid[s : e] = True

    # 6. Fallback: time-aware linear interpolation
    valid = np.isfinite(out)
    if not np.all(valid):
        valid_times = times[valid]
        valid_vals = out[valid]
        out[:] = np.interp(times, valid_times, valid_vals)


def impute(
    observation_values: np.ndarray,
    observation_times: np.ndarray,
    prediction_mask: np.ndarray,
) -> np.ndarray:
    """
    Impute values at positions where prediction_mask is True.
    observation_values: (T, D) array, NaN at unobserved and holdout.
    observation_times: (T,) timestamps for each time step.
    prediction_mask: (T, D) True at positions we must predict.

    Returns:
        Full array same shape (T, D), with all NaNs filled (imputed).
    """
    out = np.array(observation_values, copy=True, dtype=np.float64)
    times = np.asarray(observation_times, dtype=np.float64)

```

```
    if out.ndim == 1:
        _impute_1d(out, times)
        return out

    T, D = out.shape
    if T == 0 or D == 0:
        return out

    for d in range(D):
        _impute_1d(out[:, d], times)

    return out
```

G. Hyperparameters and System Prompts

All experiments use OpenEvolve (Sharma, 2025). Across tasks, we used the same LLM backbone and the same core evolutionary setup: a weighted ensemble with `gpt-5-mini-2025-08-07` (weight 0.8) and `gpt-5-2025-08-07` (weight 0.2), temperature 0.7, `max_tokens`=16,000, model timeout 120s, diff-based evolution, `max_code_length`=20,000, archive size 20, population size 50, and MAP-Elites with 3 islands. All runs used `elite_selection_ratio`=0.2 and `similarity_threshold`=0.99; `exploitation_ratio` was 0.7. The evaluator used `parallel_evaluations`=5 and `cascade_thresholds`=$\{0.0\}$ in all three tasks. Evaluator timeouts were 60s for point processes and MJP, and 3600s for imputation.

Point processes. The point-process task optimized a zero-shot heuristic for next-event time and mark prediction. The score was

$$\text{score} = \text{accuracy} - \lambda \cdot \text{RMSE}_{\Delta t}, \qquad \lambda = 1.$$

The exact system prompt was:

```
You are an expert statistician and algorithm designer for temporal point processes and
↪  marked event sequences.
Your task is to improve a zero-shot Python heuristic that predicts the next event time and
↪  mark (type) given
a batch of sequence histories and a context pool of sequences.

CONSTRAINTS (strict):
- Do NOT use PyTorch, TensorFlow, JAX, or any other heavy ML framework.
- Use only standard library Python and NumPy.
- All code must stay inside the EVOLVE-BLOCK and implement the function
↪  predict_next_events with this exact signature:
  predict_next_events(target_history_times, target_history_marks, context_times,
  ↪  context_marks, num_classes)
- The function must return a tuple (predicted_next_times, predicted_next_marks), both 1D
↪  NumPy arrays of length batch_size.
```

Markov jump processes. For MJP, the evolved program estimated the initial distribution and continuous-time rate matrix from discretely observed trajectories. The score combined the cross-entropy of the initial distribution and the RMSE of the off-diagonal rate entries,

$$\text{score} = -\text{CE} - \text{RMSE}_Q.$$

The evaluator used a cascade with threshold 0.0 before running the full metric computation. The exact system prompt was:

```
You are an expert in stochastic processes, Markov Jump Processes (MJP), and numerical
↪  algorithms.
Your task is to write a pure Python/NumPy heuristic that estimates the continuous-time
transition rate matrix (intensity matrix Q) and the initial probability distribution.

CRITICAL DOMAIN KNOWLEDGE:
1. The data consists of *discrete snapshots* (recordings at specific times), NOT exact
↪  jump times. You do not see every state transition. Between `observation_grid[i]` and
↪  `observation_grid[i+1]`, zero, one, or multiple hidden jumps could have occurred.
```

```
2. The data is noisy. The `observation_values` might contain measurement errors.

Return NO text outside the python code block. Do NOT use torch. Use only numpy.
```

Imputation. For imputation, the evolved function filled held-out entries in reduced time series, and the score was the negative mean MAE across datasets,

$$\text{score} = -\text{mean MAE}.$$

The config used the datasets BEIJING, ITALY, GUANGZHOU, PEMS, PEDESTRIAN, SOLAR, ETT_H1, ELECTRICITY, MOTIONCAPTURE_PCA, and MOTIONCAPTURE_NOPCA. The standard evaluation holdout fraction was 10% with seed 42 and a global cap of 1000 timesteps; for the two MotionCapture datasets, the holdout fraction was increased to 20% and averaged over 10 folds. The exact system prompt was:

```
You are an expert in time series analysis, signal processing, and advanced imputation
↪  algorithms.
Your task is to improve a zero-shot Python heuristic that predicts values at positions
↪  indicated by a mask.
Your algorithm should capture both complex oscillatory datasets but also work well with
↪  simple datasets.
The data is REDUCED: only originally-observed positions are kept. The only "missing"
↪  values are the eval holdout (NaN). Note that these missing values can be windows of
↪  missing values or random missing values.
The function receives observation_values (NaN at holdout), observation_times (exact
↪  timestamps for each position), and prediction_mask (True at positions to predict).
Note that it could also be that all elements for one channel are NaN so you need to handle
↪  this case as well.
You are allowed to use numpy. If possible, try to use vectorized operations for
↪  efficiency.

ALGORITHMIC DIVERSITY & INSPIRATION:
CONSTRAINTS (strict):
- All code must stay inside the EVOLVE-BLOCK and implement the function impute with this
↪  exact signature:
  impute(observation_values, observation_times, prediction_mask)
- observation_values: reduced 1D array, with NaN at holdout positions.
- observation_times: 1D array of timestamps for each position (same length as
↪  observation_values).
- prediction_mask: boolean array, True at positions we must predict.
- The function must return an array of the same shape as observation_values with all NaNs
↪  filled (imputed).
- Lower MAE at the predicted positions is better.
```