EVIL: EVOLVING INTERPRETABLE ALGORITHMS FOR ZERO-SHOT INFERENCE ON EVENT SEQUENCES AND TIME SERIES WITH LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce `EVIL` (**EV**olving **I**nterpretable algorithms with **L**LMs), an approach that uses LLM-guided evolutionary search to discover simple, interpretable algorithms for dynamical systems inference. Rather than training neural networks on large datasets, `EVIL` evolves pure Python/NumPy programs that perform zero-shot, in-context inference across datasets. We apply `EVIL` to three tasks: next-event prediction in temporal point processes, rate matrix estimation for Markov jump processes, and time series imputation. In each case, a single evolved algorithm generalizes across all evaluation datasets without per-dataset training, like an amortized inference model. A variant evolved on synthetic data alone does the same, having seen none of those datasets during the search. To our knowledge, this is the first work showing that LLM-guided program evolution can discover a single compact inference function for these problems. The picture is regime-dependent: on long-horizon event prediction the evolved programs have the best optimal-transport distance of all classical models and are competitive with the neural and foundation models. On Markov jump process rate estimation they beat every classical estimator, and a pretrained foundation model as well, except under heavy observation noise. On imputation they match the strongest classical interpolation at point-wise missingness, but stay behind the neural baselines on missing windows. They also outperform classical methods built on similar concepts in nearly every comparison, so the search seems to derive new procedures rather than restating known ones. Inference is roughly two orders of magnitude faster than the neural baselines, and every discovered algorithm stays fully interpretable.

1 INTRODUCTION

Recent work on dynamical systems has been dominated by deep learning methods, including temporal point processes (Mei & Eisner, 2017; Mei et al., 2022; Zeng et al., 2024), Markov jump processes (Seifner & Sánchez, 2023; Berghaus et al., 2024), and time series imputation (Tashiro et al., 2021; Seifner et al., 2025b). However, these approaches share several limitations: (i) they require large curated training sets and are often expensive to train and run; (ii) they produce opaque predictions, making it difficult to understand *why* a particular prediction was made; and (iii) they often generalize poorly to distribution shifts between training and test data, or to output dimensions not seen during training.

We propose a fundamentally different strategy: Instead of learning parameters of a neural network from data, we use LLM-guided evolutionary search to *discover* simple, interpretable algorithms that solve these tasks directly. Our approach, `EVIL` (**EV**olving **I**nterpretable algorithms with **L**LMs), leverages AlphaEvolve-style program evolution (Novikov et al., 2025) to iteratively refine Python functions that take in context data and produce predictions using only NumPy operations.

Our main contributions are threefold. (i) We introduce a new *application* of LLM-guided program evolution. The search paradigm itself is not ours: it was established by FunSearch, AlphaEvolve, OpenEvolve and LLM-SR (Romera-Paredes et al., 2023; Novikov et al., 2025; Sharma, 2025; Shojaee et al., 2024), and we use it essentially as published. What is new is the target of the search: not a solution to a single fixed instance or dataset, but one simple Python/NumPy procedure per task. It

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

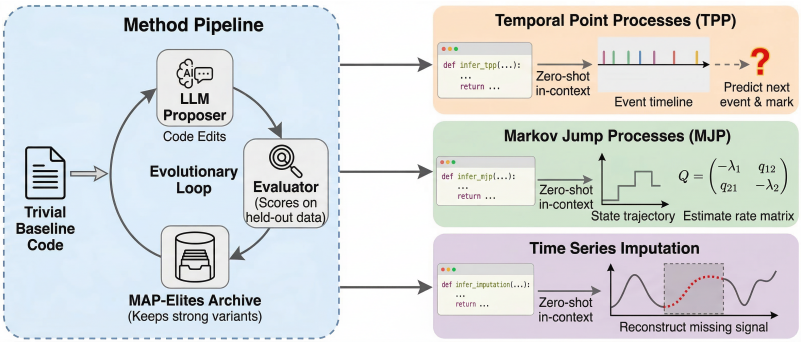


Figure 1: Overview of the EVIL approach. The same EVIL evolutionary procedure is applied separately to temporal point processes, Markov jump processes, and time-series imputation, yielding one interpretable Python/NumPy inference function per task that generalizes across datasets in a zero-shot, in-context manner.

takes context observations, predicts directly at test time without per-dataset retraining, and generalizes to arbitrary numbers of dimensions. To the best of our knowledge, this is the first use of AlphaEvolve-style program evolution in this way. (ii) We provide empirical evidence that such amortized functions actually exist for several nontrivial problem classes. Across temporal point-process prediction, Markov jump process inference, and time-series imputation, the evolved programs generalize across datasets in a zero-shot, in-context manner (for the variant evolved on synthetic data alone, without ever having seen those datasets during the search). They are often already competitive with substantially more complex learned baselines, while being faster, cheaper, and fully interpretable. It was not previously clear that such compact, reusable inference functions could be found at all across diverse datasets, nor that an LLM-guided evolutionary process could discover them automatically. We also compare against classical statistical methods that build on similar concepts, run under the same protocol and data access, and the evolved programs outperform them in nearly every comparison. (iii) We use these three applications to illustrate a broader picture: EVIL is not a collection of task-specific tricks, but a general recipe for searching over simple inference procedures that exploit structure in contextual data.

The main practical advantages of EVIL are:

- **Competitive performance, stated per regime**: the EVIL variants collectively improve on every classical model in long-horizon point-process OTD and achieve the best OTD of all methods on three of five datasets; it beats every classical MJP rate estimator at both noise levels and a pretrained foundation model at the lower one; it matches the best classical interpolator on point-wise imputation and stays behind the neural baselines on windowed imputation (Sections 4–6).
- **Generalization**: A single algorithm per task works across all datasets without per-dataset finetuning. EVIL(s), evolved on synthetic data alone, generalizes to unseen datasets with far more categories than seen during evolution (Table 2) and is more robust under distribution shift (Figure 2).
- **Interpretability and low cost**: Every evolved algorithm is a readable Python function (Appendix J), found in well under an hour for less than $1 of API cost, and inference needs only basic array operations (Table 19).[1]

2 Related Work

2.1 LLM-Guided Program Evolution

AlphaEvolve (Novikov et al., 2025) introduced an evolutionary framework where LLMs propose code modifications that are scored by automated evaluators, achieving advances in algorithm design and

[1]The source code of the discovered algorithms can be found at https://anonymous.4open.science/r/EVIL_ICLR_attachment-372F.

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.

2.2 Foundation Models and In-Context Inference

Recent work on dynamical systems has explored synthetic pretraining and amortized in-context inference as a general alternative to fitting a new neural network for each dataset. In the broader in-context-learning literature, prior-fitted neural networks learn to approximate predictive posteriors directly from context-target examples (Müller et al., 2022). Within dynamical systems, the Foundation Inference Model (FIM) line instead emphasizes zero-shot estimation of system-defining operators or trajectories from contextual observations, covering rate-matrix inference in Markov jump processes (Berghaus et al., 2024), ordinary (Mauel et al., 2026) and stochastic (Seifner et al., 2025a) differential equations, imputation in dynamical systems (Seifner et al., 2025b), and conditional-intensity inference in temporal point processes (Berghaus et al., 2026). These works form the main foundation-model context for our setting; the domain-specific methods that require training on each individual dataset are introduced in the corresponding subsections of Sections 4–6. Very recently, Zhang & Gilpin (2026) proposed *context parroting*, a simple baseline that copies relevant patterns from the input context. It is surprisingly competitive with foundation models in chaotic time series forecasting, reinforcing the message that simple, interpretable baselines can challenge trained models.

3 Methodology

3.1 Problem Setup

In each of our three applications, the task can be framed as: given a set of context observations and a target query, produce a prediction using a pure Python/NumPy function. This means predicting the next event from a history and a pool of context sequences for point processes, estimating a rate matrix from discretely observed trajectories for MJPs, and filling in missing values of a partially observed time series for imputation. In all cases, the function must work across datasets without modification, so it operates 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. Stated without the terminology, the user supplies four things: an interface, a trivial implementation of it, a problem description for the prompt, and a scoring function. The loop then alternates between asking the LLM to patch a program drawn from the database and scoring the result. The highest-scoring program at the end of the budget becomes the final one and is never modified again. Appendix I turns this into a recipe for applying `EVIL` to a new problem. 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 K. 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 subsample from the existing training split of each dataset. The subsample is carved out of the *training* split only, so no score computed on a validation or test split ever entered the search. The test splits are touched exactly once, to produce the numbers reported here; Appendix B states the construction for each variant and each task. This also 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 subsample (e.g.,

accuracy and RMSE for point processes, cross-entropy and RMSE for MJPs, MAE for imputation). We report two variants: `EVIL`, evolved on subsamples of the target datasets, and `EVIL(s)`, evolved on synthetic data only, where the "s" stands for the synthetic prior it is evolved from.

We distinguish three senses of "zero-shot". All `EVIL` programs are *test-time zero-shot*: they contain no dataset-specific parameters, so no fitting happens when the final program runs. `EVIL` is *target-subsample evolved*: subsamples of the target datasets guided the search, so it is best read as an amortized algorithm fitted once across datasets by program search rather than per dataset by gradient descent. `EVIL(s)` is *strict zero-shot*: no real target data was involved at any point, so it carries our zero-shot and cross-dataset generalization claims. Every table caption states which regime each row belongs to.

Reported runtimes are the inference cost of the final program, excluding the one-off search (15 to 40 minutes and less than $1 of API cost per task). 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, the choice of evolution LLM (Appendix E.5), and a substantially longer run are reported in Appendix E.

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 C 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, and the evolutions (100 iterations) took about 15 minutes of wall-clock time. We evolve two variants: `EVIL`, evolved on small subsamples of the target datasets in Table 1, and `EVIL(s)`, evolved only on subsamples of the synthetic data from Berghaus et al. (2026). This makes the comparison to FIM-PP (Berghaus et al., 2026) particularly natural, since it too is built around synthetic pretraining and is then evaluated either zero-shot or after finetuning, though on substantially more data overall (14.4 million events, against 72k for `EVIL(s)`).

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. Since the evolved program is itself a simple statistical procedure, we additionally run the standard classical point-process models under the same protocol and data access as `EVIL`. These are a homogeneous Poisson process with a constant event rate (Daley & Vere-Jones, 2003); a renewal process with gaps drawn from a fitted distribution (Cox & Lewis, 1966); a Markov-renewal (semi-Markov) process, in which the next mark comes from a transition table and the waiting time depends on that transition (Pyke, 1961); the Autoregressive Conditional Duration (ACD) model, in which each waiting time depends on recent ones as GARCH models volatility (Engle & Russell, 1998); and a self-correcting process, in which each event makes the next one less likely (Isham & Westcott, 1979). We also report the un-evolved seed.

Table 1: Long-horizon prediction (N=5), OTD and sMAPE$_{\Delta t}$ (both lower is better) for the best neural baseline (CDiff; best on three of five datasets, all neural baselines in Table 10), classical Hawkes, the two strongest added classical models (all: Table 6), FIM-PP (FIM-PP(zs): zero-shot, FIM-PP(f): finetuned), the un-evolved seed, and EVIL. Neural and FIM-PP numbers from Zeng et al. (2024); Berghaus et al. (2026); all other rows run by us with the same protocol and data access. Bold: best displayed value. EVIL is target-subsample evolved, EVIL(s) is strict zero-shot.

	TAXI		TAOBAO		STACKOV.		AMAZON		RETWEET	
Method	OTD	sM	OTD	sM	OTD	sM	OTD	sM	OTD	sM
CDiff (neural)	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
Markov-renewal	4.15	**67.3**	11.47	168.6	11.92	92.6	11.78	54.0	15.65	87.5
ACD	4.26	69.5	10.81	164.8	11.72	**81.5**	11.92	71.8	15.74	**79.0**
FIM-PP(zs)	6.77	74.9	15.95	168.3	11.52	93.3	11.12	119.1	15.75	98.7
FIM-PP(f)	4.08	71.1	13.17	146.9	**10.35**	86.4	10.03	78.7	15.65	83.0
Seed (un-evolved)	11.51	68.1	11.73	**114.9**	12.34	87.7	11.68	**51.9**	16.03	111.5
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: Generalization to MIMIC-II (Lee et al., 2011), an unseen dataset with 75 marks (vs. max 22 during evolution). Baselines from Panos (2024). EVIL is target-subsample evolved and EVIL(s) is strict zero-shot; neither was ever scored on this dataset.

Method	RMSE	Acc.	Method	RMSE	Acc.
THP (Zuo et al., 2021)	1.00	84.88	IFTPP (Shchur et al., 2020)	0.74	85.13
SAHP (Zhang et al., 2020)	1.49	83.61	DTPP (Panos, 2024)	**0.72**	85.51
A-NHP (Mei et al., 2022)	1.00	84.81	**EVIL(s)**	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 J) 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. On OTD, which scores event timing, marks and counts jointly, the two EVIL variants together improve on every classical model (Table 6) and attain the best value on three of the five datasets. On sMAPE$_{\Delta t}$ they do not. That metric is a relative error on the inter-event times alone, so it rewards conservative, nearly constant gaps. It is usually won by a classical model or by the un-evolved seed, which is among the weakest on OTD. The evolved program also reuses ingredients from the classical models, a mark-transition table as in Markov-renewal processes and recency-weighted gaps as in ACD, though no single classical model combines them as it does (Appendix D.1).

On this benchmark, FIM-PP evaluation took 964s on an A100 GPU and EVIL 5s on a single Apple M3 CPU core; the classical models are of comparable speed to EVIL (Table 19), so the speed argument is one against the neural baselines only.

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

Generalization. On next-event (N=1) prediction (Table 7 in Appendix D), both EVIL variants match the best neural baselines in accuracy on TAXI while substantially improving sMAPE, and EVIL(s) reaches that accuracy from synthetic data alone, where FIM-PP(zs) reaches only 52%. Since the frequent mark alternations of TAXI (Figure 3) are not covered well by the synthetic data of Berghaus et al. (2026), the evolved algorithm generalized beyond its own synthetic prior as well. On MIMIC-II (Lee et al., 2011), never seen during evolution and with 75 mark types instead of at most 22, EVIL reaches 87.8% accuracy, above all neural baselines trained on it specifically (Table 2).

Table 3: Rate-matrix recovery on 300 held-out synthetic six-state MJPs (300 paths each), unseen by the search and by FIM-MJP (applied zero-shot): RMSE of the off-diagonal rates (lower is better), $\pm$ one standard deviation over MJPs; 95% bootstrap intervals of the means separate all adjacent rows. Four reversible estimators agree to within 10^{-5} and share one row. `EVIL(s)` is strict zero-shot.

Method	1% noise	10% noise
Seed (un-evolved)	0.624 ± 0.101	0.588 ± 0.110
Reversible MSM $\equiv$ `rate_matrix` (KL, CVE, truncated log)	0.568 ± 0.124	0.950 ± 0.213
Non-reversible MSM $\equiv$ matrix logarithm	0.491 ± 0.148	0.898 ± 0.234
`rate_matrix` (pseudo-generator)	0.376 ± 0.069	0.539 ± 0.074
CT-MLE (exposure times and transition counts)	0.327 ± 0.075	0.329 ± 0.075
FIM-MJP (pretrained foundation model, zero-shot)	0.254 ± 0.077	$\mathbf{0.260 \pm 0.070}$
`EVIL(s)`	$\mathbf{0.207 \pm 0.066}$	0.302 ± 0.049

Neural TPP models typically fix their output dimension to the training mark count, whereas `EVIL` handles arbitrary mark counts through 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 on a finite state space $\{1, \ldots, K\}$, 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, a challenging inverse problem since exact jump times are unobserved. Appendix F gives further background, the datasets, the metrics, and the 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, and the evolutions (100 iterations) took about 14 minutes of wall-clock time. Since ground-truth rate matrices are only available for synthetic data, `EVIL(s)` 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 substantially more complex architectures with millions of parameters. FIM-MJP is pretrained, so we run it zero-shot on the held-out generators of Table 3. NeuralMJP is fit per target system, so appearing there would mean training 300 models per noise level; where both are reported, FIM-MJP is the stronger anyway (Appendices F.7 and F.8). We additionally compare against the classical estimators for this problem, in three groups. The first are Markov state models (MSMs) (Prinz et al., 2011); we use every applicable estimator of the `deeptime` library (Hoffmann et al., 2022) (maximum-likelihood, Bayesian and Koopman/VAMP, each assuming equilibrium and not), together with the matrix-logarithm embedding that turns a fitted transition matrix into a generator (Israel et al., 2001). The second is `deeptime`'s continuous-time generator estimator `rate_matrix`, in all four of its variants. The third is the direct continuous-time maximum-likelihood estimator (CT-MLE), which divides the observed $i \to j$ transition counts by the total time spent in state i (Albert, 1962). This is the matched non-neural reference here, since it forms its estimate from the same statistics that the evolved program works from. We also report the un-evolved seed. Appendix F.5 lists them and how they differ in their treatment of time discretization and reversibility.

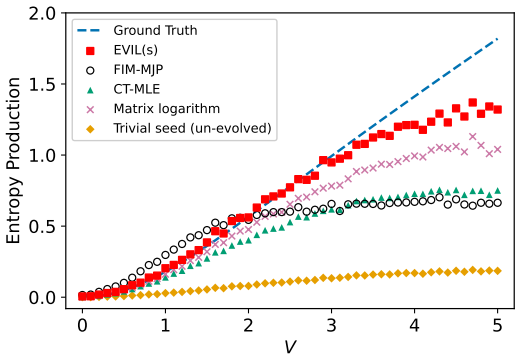


Figure 2: Entropy production of the DFR against the potential strength V, a stress test that gets harder as the rates grow more dissimilar. `EVIL(s)` tracks the ground truth across the whole range, while FIM-MJP, CT-MLE and the matrix logarithm all saturate once the asymmetry grows, and the un-evolved seed recovers almost none of the current. All six series come from one matched sweep on bit-identical paths; the reversible estimators are not shown, since their entropy production is exactly zero by construction (Appendix F.9).

5.3 Discovered Algorithm

The evolved MJP heuristic (see Pseudocode 3 and Python Algorithm 3 in Appendix J) 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

Table 3 reports off-diagonal rate recovery on held-out synthetic MJPs, the setting in which ground truth is available. `EVIL(s)` is more accurate than every classical estimator at both noise levels, and is among the cheapest methods to run (Table 19). Against FIM-MJP the picture splits by noise level: `EVIL(s)` is ahead at 1% and behind at 10%, with non-overlapping bootstrap intervals in both directions. FIM-MJP is close to invariant under the tenfold increase in noise, which its pretraining distribution explicitly contains, whereas `EVIL(s)` degrades: its noise handling is a few discovered corrections rather than a learned invariance. CT-MLE is the closest classical reference, since it works from the same exposure times and transition counts, and `EVIL(s)` improves on it at both noise levels, by a wide margin at 1%. What the search changed is how those statistics are formed: it caps the exposure time, discards single-observation excursions before they count as dwells, and weights destination counts by recency. Applying the evolved program's smoothing constants to CT-MLE changes its RMSE by less than 10^{-3}, so the improvement comes from these changes to the estimator itself and not from smoothing.

FIM-MJP's robustness does not extend beyond label noise. The Discrete Flashing Ratchet (DFR) (Ajdari & Prost, 1992) is a 6-state non-equilibrium system whose generator is fixed by three physical parameters (V, r, b) and which sustains an irreversible current. There, `EVIL(s)` recovers those parameters well (Appendix F.7) and tracks the ground-truth entropy production across the whole range of potential strengths (Figure 2), whereas FIM-MJP and the classical estimators all saturate from $V \approx 3$ on, and the reversible estimators cannot represent a current at all.

On real-world data (ion-channel recordings (Gazzarrini et al., 2006) and molecular dynamics (Mironov et al., 2019)) no ground truth exists, and the only metric is the Hellinger distance (Hellinger, 1909) between simulated and observed occupancy histograms. There, `EVIL(s)` is ahead of both learned models on three of the four datasets by clear margins, but the classical estimators match it to within the noise of the metric, and on PFold no method is distinguishable at all. We treat this as a sanity check rather than as evidence, since the metric sees only state occupancy and every MSM variant scores essentially the same as `EVIL(s)` (Appendix F.8).

6 Time Series Imputation

6.1 Background

Time series imputation aims to reconstruct missing values in partially observed multivariate time series. In many applications the observed sequences are not arbitrary collections of numbers, but

Table 4: MAE on 8 datasets with 50% point-wise missing. `EVIL` was evolved on training-split subsamples of the first four (target-subsample evolved) and is strict zero-shot on the other four; `EVIL(s)` is strict zero-shot on all eight. Neural and foundation-model results from Fang et al. (2023); Du et al. (2024); Seifner et al. (2025b); classical rows measured by us, reproducing TSI-Bench to within 0.01. Best in bold, ties jointly.

	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
LOCF (seed)	.258	.346	3.00	.578	.356	.927	.422	.140
Linear interpolation	.152	.204	**2.09**	.338	.242	.409	.267	**.078**
kNN motif retrieval	.153	.204	2.23	.339	.242	.520	.268	.079
`EVIL`	.152	.204	**2.09**	.338	.242	.413	.267	**.078**
`EVIL(s)`	.151	**.202**	2.11	.326	.238	**.384**	.261	.079

partial observations of a trajectory generated by a dynamical system. Both observed and missing values then lie on a latent time-evolving process, so imputation amounts to inferring the trajectory that best explains the observations and reading off the unobserved values. We consider two settings: point-wise missing patterns (random individual values are missing) and windowed 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 dataset variants: 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. Within the 6 evolved-on variants, the scoring targets are 10% of the observed entries, disjoint from the entries each benchmark evaluates on (Appendix B). Appendix G summarizes these datasets and their sources. The evolutions (100 iterations) took about 37 minutes wall-clock time. We additionally evolve **`EVIL(s)`** on synthetic data only. Each one-dimensional signal is drawn from a shared family of functions (random low-degree polynomials, Gaussian-process samples with RBF, Matérn and periodic kernels, and low-frequency Fourier series), and is then corrupted with either point-wise or windowed missingness, independently of the function. The family is fixed in advance from the generative story of Seifner et al. (2025b) and is not tuned to the benchmarks; Appendix G.3 gives it in full. The final program is selected on a held-out synthetic split and evaluated on the real datasets only once.

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., 2025b) (foundation model pretrained on 2M+ ODE solutions). All baselines except FIM-ℓ require per-dataset training. For Motion Capture, we additionally report the cubic spline and Savitzky–Golay filtered spline (Spline(F)) baselines of Seifner et al. (2025b). We also run classical imputers ourselves under the same protocol: global per-channel mean and median fills as in TSI-Bench (Du et al., 2024), last observation carried forward (LOCF, the un-evolved seed of our search), linear and cubic spline interpolation, and a k-nearest-neighbour motif-retrieval baseline that copies the continuation of the earlier segment whose context best matches the context before a gap; since motif retrieval is the distinctive component of the evolved program, this shows how much of the gain that mechanism alone explains. The comparison on all ten dataset variants is in Appendix G.4.

Table 5: MAE on Motion Capture (20% windowed missing), mean ± std over 43 trajectories. Neural results from Seifner et al. (2025b); classical rows measured by us (our Spline matches their published value, their Spline(F) did not reproduce; Appendix G.4). `EVIL` is target-subsample evolved, `EVIL(s)` is strict zero-shot.

Model	MC (PCA)	MC (No PCA)	Model	MC (PCA)	MC (No PCA)
LatentODE	$\mathbf{1.66 \pm 0.99}$	–	Spline	3.36 ± 1.16	4.21 ± 1.42
FIM-ℓ	1.77 ± 0.63	$\mathbf{1.61 \pm 0.45}$	Spline(F)	3.37 ± 1.15	4.21 ± 1.41
LOCF (seed)	6.27 ± 2.64	6.38 ± 2.70	kNN motif retrieval	3.34 ± 1.56	3.68 ± 1.76
Linear interpolation	3.96 ± 1.22	4.07 ± 1.32	**`EVIL`**	2.98 ± 2.03	3.34 ± 2.07
			`EVIL(s)`	3.66 ± 1.16	3.83 ± 1.22

6.3 Discovered Algorithm

The evolved imputation heuristic (see Pseudocode 4 and Python Algorithm 4 in Appendix J) 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.* The synthetic-prior variant **`EVIL(s)`** (see Pseudocode 5 and Python Algorithm 5 in Appendix J) instead evolves a gap-adaptive interpolation scheme, filling short gaps by linear interpolation and long gaps by a local monotone PCHIP spline fitted to the observed points on each side.

6.4 Results and Analysis

Table 4 shows point-wise imputation results. `EVIL` is competitive on all 8 datasets, including the 4 held out from the search entirely, and `EVIL(s)` matches it while being the best of all methods on two of them. The classical rows show what the table does not: on point-wise missingness `EVIL` is essentially plain linear interpolation, agreeing with it at the reported precision on all but Solar, where interpolation is marginally better. That is consistent both with TSI-Bench (Du et al., 2024) and with the evolved program itself, which falls back to time-aware linear interpolation whenever a gap is short, and at 50% missingness almost every gap is one to three samples long.

Table 5 shows windowed imputation on Motion Capture, where `EVIL` roughly halves the error of the seed it started from and is the best method without per-dataset neural training. FIM-ℓ and LatentODE remain better, which suggests that contiguous windows benefit from the learned dynamical priors FIM-ℓ acquires during pretraining on ODE solutions, though the variances are high across all methods. `EVIL(s)` improves on the seed and on linear interpolation here as well and is comparable to the cubic spline, without reaching `EVIL`. The mechanism behind this is the *motif* retrieval (Lin, 2002) described above, which is enough to reconstruct structured oscillations in some cases (Figure 6). The retrieval-only baseline beats both spline variants on long gaps but hurts point-wise data (Appendix G.4), so what the search contributed is the gate that decides when *not* to retrieve. The concurrent *context parroting* baseline of Zhang & Gilpin (2026) also relies on a motif-based strategy, suggesting that motif retrieval may be a naturally strong inductive bias that human-designed and LLM-discovered methods converge on independently.[2]

7 Conclusion

We explored whether LLM-guided evolutionary search can discover explicit, interpretable algorithms for dynamical systems inference; across three tasks it found compact Python functions that perform zero-shot inference without per-dataset training. They trail deep networks where rich learned representations matter, such as windowed imputation, but are competitive on standard benchmarks, ahead of the classical models on long-horizon event prediction and MJP rate estimation, orders of magnitude faster than the neural baselines, and fully transparent. We hope `EVIL` serves as a cheap, grounded baseline for checking when the complexity of a new model is justified. Appendix A discusses why `EVIL` works and where it falls short.

[2]That paper appeared after the knowledge cutoff of the LLM, so the strategy was found independently.

AI USE STATEMENT

Large language models are a core component of the method studied in this paper: `EVIL` uses them to propose program modifications during the evolutionary search, and the models and prompts used for this are documented in Section 3 and Appendix K. Beyond this, we used generative AI tools for coding and literature search, and to assist with writing the paper. Figure 1 was generated with a generative AI tool and subsequently refined by a human. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The evolution protocol and the construction of the evolution and evaluation splits are described in Appendix B, the hyperparameters and system prompts in Appendix K, and all discovered algorithms are listed in full in Appendix J. The source code of the discovered algorithms can be found at https://anonymous.4open.science/r/EVIL_ICLR_attachment-372F.

REFERENCES

Armand Ajdari and Jacques Prost. 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.

Arthur Albert. Estimating the infinitesimal generator of a continuous time, finite state markov process. *The Annals of Mathematical Statistics*, 33(2):727–753, 1962.

Emmanuel Bacry, Martin Bompaire, Philip Deegan, Stéphane Gaïffas, and Søren V. Poulsen. 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.

David Berghaus, Kostadin Cvejoski, Patrick Seifner, Cesar Ojeda, and Ramses J Sanchez. 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.

David Berghaus, Patrick Seifner, Kostadin Cvejoski, Cesar Ojeda, and Ramses J Sanchez. 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.

Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. BRITS: Bidirectional recurrent imputation for time series. *Advances in Neural Information Processing Systems*, 31, 2018.

David R. Cox and Peter A. W. Lewis. *The Statistical Analysis of Series of Events*. Methuen, 1966.

Daan Crommelin and Eric Vanden-Eijnden. Data-based inference of generators for markov jump processes using convex optimization. *Multiscale Modeling & Simulation*, 7(4):1751–1778, 2009.

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

E. Brian Davies. Embeddable markov matrices. *Electronic Journal of Probability*, 15:1474–1486, 2010.

Prathamesh Deshpande, Kamlesh Marathe, Abir De, and Sunita Sarawagi. 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.

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

Wenjie Du, Jun Wang, Linglong Qian, Yiyuan Yang, Zina Ibrahim, Fanxing Liu, Zepu Wang, Haoxin Liu, Zhiyuan Zhao, Yingjie Zhou, et al. Tsi-bench: Benchmarking time series imputation. *arXiv preprint arXiv:2406.12747*, 2024.

Robert F. Engle and Jeffrey R. Russell. Autoregressive conditional duration: A new model for irregularly spaced transaction data. *Econometrica*, 66(5):1127–1162, 1998.

Shikai Fang, Qingsong Wen, Yingtao Luo, Shandian Zhe, and Liang Sun. Bayotide: Bayesian online multivariate time series imputation with functional decomposition. *arXiv preprint arXiv:2308.14906*, 2023.

Vincent Fortuin, Dmitry Baranchuk, Gunnar Rätsch, and Stephan Mandt. GP-VAE: Deep probabilistic multivariate time series imputation. In *Proceedings of AISTATS*, pp. 1651–1661, 2020.

Sabrina Gazzarrini, Ming Kang, Svetlana Epimashko, James L Van Etten, Jack Dainty, Gerhard Thiel, and Anna Moroni. Chlorella virus MT325 encodes water and potassium channels that interact synergistically. *Proceedings of the National Academy of Sciences*, 103(14):5355–5360, 2006.

Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025.

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

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

Markus Heinonen, Cagatay Yildiz, Henrik Mannerström, Jukka Intosalmi, and Harri Lähdesmäki. Learning unknown ode models with gaussian processes. In *International Conference on Machine Learning*, pp. 1959–1968. PMLR, 2018.

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

Moritz Hoffmann, Martin Scherer, Tim Hempel, Andreas Mardt, Brian de Silva, Brooke E. Husic, Stefan Klus, Hao Wu, Nathan Kutz, Steven L. Brunton, and Frank Noé. Deeptime: a python library for machine learning dynamical models from time series data. *Machine Learning: Science and Technology*, 3(1):015009, 2022.

Brooke E. Husic, Nicholas E. Charron, Dominik Lemm, Jiang Wang, Adrià Pérez, Maciej Majewski, Andreas Krämer, Yaoyi Chen, Simon Olsson, Gianni de Fabritiis, Frank Noé, and Cecilia Clementi. Coarse graining molecular dynamics with graph neural networks. *The Journal of Chemical Physics*, 153(19):194101, 2020.

Valerie Isham and Mark Westcott. A self-correcting point process. *Stochastic Processes and their Applications*, 8(3):335–347, 1979.

Robert B. Israel, Jeffrey S. Rosenthal, and Jason Z. Wei. Finding generators for markov chains via empirical transition matrices, with applications to credit ratings. *Mathematical Finance*, 11(2): 245–265, 2001.

John D. Kalbfleisch and Jerald F. Lawless. The analysis of panel data under a markov assumption. *Journal of the American Statistical Association*, 80(392):863–871, 1985.

Joon Lee, Daniel J Scott, Mauricio Villarroel, Gari D Clifford, Mohammed Saeed, and Roger G Mark. 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.

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

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

Haitao Lin, Lirong Wu, Guojiang Zhao, Liu Pai, and Stan Z. Li. Exploring generative neural temporal point process. *Transactions on Machine Learning Research*, 2022.

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

Andreas Mardt, Luca Pasquali, Hao Wu, and Frank Noé. Vampnets for deep learning of molecular kinetics. *Nature Communications*, 9, 2017.

Maximilian Mauel, Johannes R Hübers, David Berghaus, Patrick Seifner, and Ramses J Sanchez. Foundation inference models for ordinary differential equations. *arXiv preprint arXiv:2602.08733*, 2026.

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

Hongyuan Mei, Guanghui Qin, and Jason Eisner. Imputing missing events in continuous-time event streams. In Kamalika Chaudhuri and Ruslan Salakhutdinov (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.

Hongyuan Mei, Chenghao Yang, and Jason Eisner. Transformer embeddings of irregularly spaced events and their participants. In *International Conference on Learning Representations*, 2022.

Vladimir Mironov, Yuri Alexeev, Vikram Khipple Mulligan, and Dmitri G. Fedorov. A systematic study of minima in alanine dipeptide. *Journal of Computational Chemistry*, 40(2):297–309, 2019.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do Bayesian inference. *arXiv preprint arXiv:2112.10510*, 2022.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 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.

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

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

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

Jan-Hendrik Prinz, Hao Wu, Marco Sarich, Bettina Keller, Martin Senne, Martin Held, John D. Chodera, Christof Schütte, and Frank Noé. Markov models of molecular kinetics: Generation and validation. *The Journal of Chemical Physics*, 134(17):174105, 2011.

Ronald Pyke. Markov renewal processes: Definitions and preliminary properties. *The Annals of Mathematical Statistics*, 32(4):1231–1242, 1961.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2023. doi: 10.1038/s41586-023-06924-6.

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

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

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

Patrick Seifner, Kostadin Cvejoski, David Berghaus, Cesar Ojeda, and Ramses J Sanchez. In-context learning of stochastic differential equations with foundation inference models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=ceCJPoZOKJ.

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

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

Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020.

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K Reddy. LLM-SR: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

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

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

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

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

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

Qiang Zhang, Aldo Lipani, Omer Kirnap, and Emine Yilmaz. Self-attentive Hawkes process. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 11183–11193. PMLR, 2020.

Yuanzhao Zhang and William Gilpin. 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.

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

Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 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.

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer hawkes process. In *International Conference on Machine Learning*, 2021.

A Discussion and Limitations

A.1 Why Does EVIL Work?

We see three reasons why compact evolved programs are competitive with much larger learned models. They are hypotheses rather than isolated effects, but each is consistent with evidence reported elsewhere in the paper.

The LLM brings prior knowledge. The LLM has extensive knowledge of statistical methods and algorithm design, which gives the search an informed starting point that a neural network would have to learn from data. The discovered programs reflect this: the evolved point-process program reuses a mark-transition table as in Markov-renewal processes and recency-weighted gap statistics as in ACD (Appendix D.1). No single classical model combines the two, and the evolved combination improves on all of them in OTD.

The programs compute statistics exactly. Evolved programs perform direct and exact statistical computations on the context data rather than approximating these operations through learned parameters. On the MJP task, `EVIL(s)` works from the same exposure times and transition counts as the continuous-time maximum-likelihood estimator; what the search changed is how these statistics are formed, and applying the evolved smoothing constants to CT-MLE changes its RMSE by less than 10^{-3} (Section 5). The advantage therefore comes from a better estimator built on the same exact statistics, not from a learned approximation of one.

Short programs favor simple solutions. Solutions that must be expressed as short programs carry an implicit bias toward simpler approaches that generalize better, reminiscent of Occam's razor and of observations made in program synthesis for mathematical discovery (Georgiev et al., 2025). The long run in Appendix E.3 is consistent with this: with eight times the search budget, the search produced a program of roughly 400 lines whose small gains on the validation set did not carry over consistently to the test set.

A.2 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 novelty of the discovered algorithms is also difficult to establish conclusively, though there is evidence for it: they outperform the classical baselines built on the same concepts in nearly every comparison, and we are not aware of these procedures appearing in this form in the literature. Beyond this, we see two further limitations. *The LLMs are opaque.* Neither their training data nor their training procedure is available, so we cannot rule out that a component of a discovered program was memorized from a source we are unaware of. *Uncertainty is not matched across all comparisons.* Repeated training runs for the foundation-model baselines are out of computational reach, so their variability cannot be compared with our evolution-run variability (Appendix E). Finally, the discovered algorithms are deterministic; finding stochastic, uncertainty-aware ones remains open.

B Evolution Protocol and Split Construction

The rule is the same in all three tasks: whatever target data the search scores on is a subsample carved out of the *training* split, the validation and test splits are never scored on, and the test split is used exactly once, to produce the numbers reported in this paper. No intermediate program was ever selected, kept, or discarded on the basis of a test score, and no program was ever re-selected after seeing a test number.

The two variants differ in which data that subsample is drawn from. For EVIL, the *target-subsample evolved* variant, it is drawn from the training split of the target datasets, so the search does see the

target distribution, though never the validation or test splits. For `EVIL(s)`, the *strict zero-shot* variant, it is drawn from synthetic data alone and no part of any target dataset enters the search at any point. Both subsamples are small relative to what is available: for point processes the search scores on 72k synthetic events where FIM-PP was pretrained on 14.4 million, and for MJPs on 7,200 synthetic processes where FIM-MJP was pretrained on 45K. Whether the search additionally sees synthetic data in the target-subsample case depends on the task, as the per-task paragraphs below spell out.

The concrete construction per task is as follows.

Point processes. The search scores on 33 synthetic datasets drawn from the FIM-PP synthetic family (Berghaus et al., 2026) (25 sampled sequences per dataset), and, for `EVIL` only, on small subsamples of the training splits of the five real datasets. The score is accuracy $-$ $\text{RMSE}_{\Delta t}$ on those sequences. MIMIC-II is not part of the search for either variant, which is why Table 2 is a transfer result for both.

Markov jump processes. Only the strict zero-shot variant exists here, because ground-truth rate matrices are available for synthetic data only, so a target-subsample variant could not be scored. The search scores on 7,200 synthetic MJPs from the same family as FIM-MJP (Berghaus et al., 2024), with state spaces of size 2 to 6, and never sees DFR, IonCh, ADP, or PFold. The held-out synthetic MJPs of Table 3 are sampled afresh from that family and were not scored on either.

Imputation. For `EVIL`, the search scores on a reduced subsample of 6 of the ten dataset variants: BEIJING, ITALY, GUANGZHOU and PEMS point-wise, and the two Motion Capture variants. Each subsample keeps only the originally observed positions, of which a further 10% is held out at random to form the scoring targets, and the score is the mean MAE over the 6 variants. PEDESTRIAN, SOLAR, ETT_H1 and ELECTRICITY are excluded from the search entirely, so those four columns of Tables 4 and 18 are zero-shot for `EVIL` as well as for `EVIL(s)`. For the TSI-Bench datasets among the six, the subsample is drawn from the *training* split, so the test split used in those tables is never seen. GUANGZHOU follows the BayOTIDE preprocessing, which provides no separate training split, so its subsample comes from the same folds that are evaluated; the 10% scoring holdout is drawn from the benchmark-*observed* positions and is therefore disjoint from the benchmark's evaluation targets, but the underlying series are the same. For `EVIL(s)`, the search scores on synthetic signals only, and the final program is selected on a held-out synthetic split.

C Additional Temporal Point Process Details

C.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.

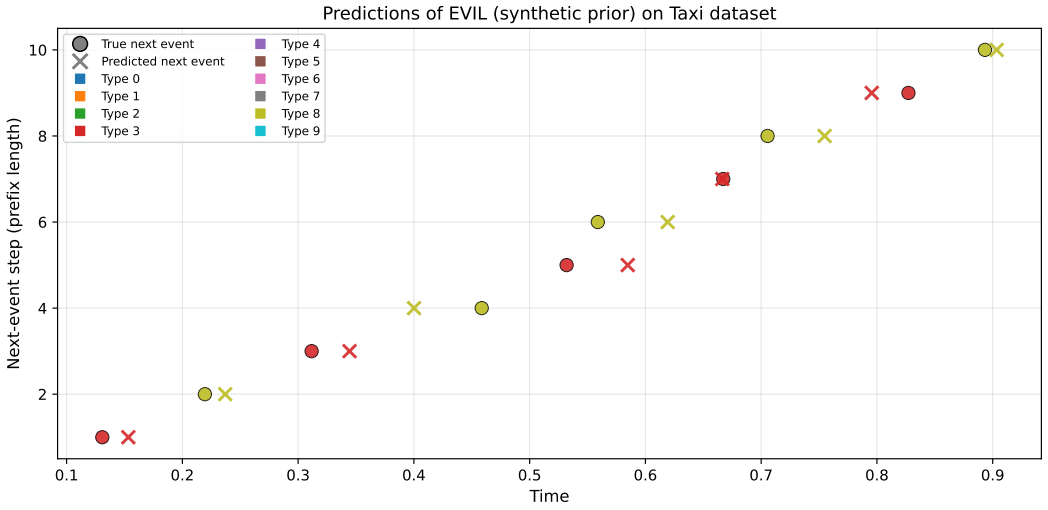


Figure 3: Illustration of mark alternation in TAXI and how the evolved next-event heuristic handles it. The dataset often alternates between marks (e.g., pick-up vs. drop-off); this pattern is not covered by FIM-PP's synthetic training data, whereas `EVIL(s)` generalizes to it from synthetic evolution.

C.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(s)` 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.

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).

C.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.

C.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.

D Long-Horizon Temporal Point Process Tables

D.1 Classical Point-Process Baselines

Table 6 reports the five classical point-process models we added, together with the HawkesEM baseline, the un-evolved seed, and the two `EVIL` variants, on the N=5 long-horizon benchmark. Every row in this table was measured by us, using the same evaluation protocol, the same test sequences, and the same data access as `EVIL`.

EVIL has the best OTD on four of the five datasets and EVIL(s) on the fifth, so against the classical models the evolved programs are ahead on that metric everywhere. On sMAPE$_{\Delta t}$ the ordering is different: ACD is best on STACKOVERFLOW and RETWEET, the Markov-renewal process on TAXI, and the un-evolved seed on TAOBAO and AMAZON. As discussed in Section 4, this is a property of the metric: sMAPE$_{\Delta t}$ only looks at inter-event times and rewards conservative, nearly constant gap predictions, which is what the seed and the gap-only classical models produce, at the cost of much worse OTD. The comparison is also the most direct evidence we have on the question of whether the search rediscovered a textbook method. The evolved program reuses components that appear in these models: a mark-transition table as in Markov-renewal processes, and recency-weighted gap statistics as in ACD. But no single classical model combines the two, and the evolved combination improves on all of them on OTD.

Table 6: Long-horizon prediction (N=5): all classical point-process baselines, the un-evolved seed, and the EVIL variants. Each cell is OTD / sMAPE$_{\Delta t}$, both lower is better. All rows were measured by us under the same protocol and data access. Best OTD per dataset in bold. EVIL is target-subsample evolved, EVIL(s) is strict zero-shot.

Method	TAXI	TAOBAO	STACKOV.	AMAZON	RETWEET
HawkesEM (Lewis & Mohler, 2011)	7.15 / 86.4	11.27 / 164.1	11.70 / 82.8	14.64 / 173.7	15.69 / 80.5
Homogeneous Poisson	11.42 / 69.3	12.68 / 168.7	12.12 / 86.6	11.92 / 73.2	15.85 / 88.0
Renewal	11.54 / 69.1	11.71 / 125.4	12.59 / 93.0	12.68 / 125.3	15.99 / 108.4
Markov-renewal	4.15 / 67.3	11.47 / 168.6	11.92 / 92.6	11.78 / 54.0	15.65 / 87.5
ACD	4.26 / 69.5	10.81 / 164.8	11.72 / 81.5	11.92 / 71.8	15.74 / 79.0
Self-correcting	11.51 / 73.6	12.45 / 151.2	12.03 / 84.1	12.09 / 90.2	15.98 / 105.8
Seed (un-evolved)	11.51 / 68.1	11.73 / 114.9	12.34 / 87.7	11.68 / 51.9	16.03 / 111.5
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

D.2 NEXT-EVENT PREDICTION

Table 7 reports next-event (N=1) prediction on TAXI and TAOBAO, discussed in Section 4.

Table 7: Next-event prediction (N=1) on two real-world datasets. Baseline results from Zeng et al. (2024); Berghaus et al. (2026). Best in bold. EVIL is target-subsample evolved, EVIL(s) is strict zero-shot.

	TAXI			TAOBAO		
Method	RMSE$_{\Delta t}$	Acc	sMAPE$_{\Delta t}$	RMSE$_{\Delta t}$	Acc	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(s)	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

D.3 FULL NEURAL AND FOUNDATION-MODEL COMPARISON

Table 8: 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. `EVIL` is target-subsample evolved, `EVIL(s)` is strict zero-shot.

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(s)`	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(s)`	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(s)`	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(s)`	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(s)`	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 9: Prediction of $N{=}10$ events on five real-world datasets. Baselines from Zeng et al. (2024); Berghaus et al. (2026). Best in bold. `EVIL` is target-subsample evolved, `EVIL(s)` is strict zero-shot.

Dataset	Method	OTD	RMSE$_e$	RMSE$_{\Delta t}$	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(s)	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(s)	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(s)	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(s)	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(s)	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 10: 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. `EVIL` is target-subsample evolved, `EVIL(s)` is strict zero-shot.

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(s)	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(s)	**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(s)	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(s)	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(s)	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

E ABLATIONS

E.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 11 shows the results for evolving only on Amazon. We find that on Amazon this dataset-specific variant substantially improves $\text{sMAPE}_{\Delta t}$, from 71.2 to 54.7 (a 23% reduction), while slightly worsening OTD, from 10.93 to 11.16. 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.

E.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 12 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. The OTD standard deviations are small relative to the gaps between methods on all five datasets (0.07 to 0.33), whereas the $\text{sMAPE}_{\Delta t}$ spread is much larger and on TAOBAO (131.8 ± 17.7) is comparable to the gaps it would have to resolve, so we do not read the sMAPE ordering of single runs as meaningful. This suggests that the method is not entirely seed-invariant, but that its OTD performance profile in particular is fairly stable.

E.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 13. In this longer run, the search produced 21 new best algorithms on the validation set, as shown in Figure 4, 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 on the test set the changes were dataset-specific rather than a consistent overall improvement: OTD improves on TAOBAO (10.94 to 9.93) and RETWEET (15.56 to 15.40) but degrades on TAXI, STACKOVERFLOW and AMAZON, and $\text{sMAPE}_{\Delta t}$ likewise moves in both directions, so the eightfold search budget did not justify the added complexity. 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 several factors combine here: the limited size of the validation set, the inherent noise in the evaluation process, and benchmark datasets that may not be challenging enough to separate 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.

E.4 IS THE EVOLUTIONARY MACHINERY NECESSARY?

The evolutionary loop is only worth its complexity if it does better than simply asking the LLM for the program many times. We tested this on the MJP task by running two arms at an equal budget of $K = 100$ LLM calls. Both start from the same seed program, use the same backbone (`gpt-oss-120b`), the same system message, and the same evaluator with the same timeout, so K is directly comparable. *Independent sampling* makes K one-shot rewrites of the seed with no feedback of any kind between them. *Evolution* runs K iterations in which each proposal is conditioned on earlier programs and their scores, as everywhere else in this paper. We report the RMSE of the estimated off-diagonal rates on the dataset the search itself scores on, since the question here is about the search rather than about generalization.

Figure 5 shows the outcome. After 100 calls, independent sampling has found a program at 0.134 and evolution one at 0.109, starting from a seed at 0.152. The right panel shows where the difference

Table 11: 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 a strong neural baseline (CDiff), FIM-PP variants, and `EVIL` variants. Baseline results from Zeng et al. (2024); Berghaus et al. (2026). FIM-PP(zs) is FIM-PP (Berghaus et al., 2026) in zero-shot mode, FIM-PP(f) is FIM-PP after finetuning on the target dataset. `EVIL` is target-subsample evolved, `EVIL(s)` is strict zero-shot.

	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-PP(zs)	6.77	74.9	15.95	168.3	11.52	93.3	11.12	119.1	15.75	98.7
FIM-PP(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 12: Long-horizon prediction (N=5) on five real-world datasets: mean $\pm$ standard deviation over five independent `EVIL` runs with different seeds, so each $\pm$ is the spread across evolution runs rather than across test instances. Only `EVIL` was re-run across seeds, which is why `EVIL(s)` does not appear here. The CDiff and FIM-PP rows are reproduced from Table 1 as a yardstick for reading that spread; FIM-PP(zs) is FIM-PP (Berghaus et al., 2026) in zero-shot mode and FIM-PP(f) is FIM-PP after finetuning on the target dataset, with baseline results from Zeng et al. (2024); Berghaus et al. (2026). Both metrics are lower is better, and bold marks the best value among the displayed rows. `EVIL` is target-subsample evolved.

	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-PP(zs)	6.77	74.9	15.95	168.3	11.52	93.3	11.12	119.1	15.75	98.7
FIM-PP(f)	**4.08**	**71.1**	13.17	146.9	**10.35**	86.4	10.03	78.7	15.65	**83.0**
EVIL	**4.08**$\pm$0.13	71.7$\pm$3.3	10.28$\pm$0.33	131.8$\pm$17.7	11.87$\pm$0.15	**85.5**$\pm$1.3	11.36$\pm$0.25	**62.0**$\pm$6.3	**15.57**$\pm$0.07	89.5$\pm$6.2

comes from. Blind rewriting fails outright on a large fraction of its calls: 29 of the 100 sampled programs did not produce a usable estimator (they crashed, returned an invalid generator, or exceeded the evaluator timeout), against 3 of 100 for evolution. The typical proposal is also much better when the LLM can see what has already been tried, with a median proposal RMSE of 0.153 against 0.271. Both effects favour the loop, and neither is visible from the best-of-K number alone. We also note the opposite side of this: the median sampled program sits at roughly seed level, so a selection step of some kind is doing part of the work in both arms.

E.5 INFLUENCE OF THE EVOLUTION LLM

To quantify how much the choice of evolution LLM matters, we re-ran the MJP evolution for 100 iterations with several different open-weight LLMs, each starting from the trivial seed (the *Seed* row in Tables 17 and 16). The resulting heuristics were evaluated with the same protocols. We ran this for the MJP task only, as a representative case; the point is robustness to the backbone, and repeating it for all three tasks would triple the cost without changing what the experiment tests. Tables 14 and 15 report the results; the `EVIL(s)` row is repeated from the main tables for reference. All tested models improve substantially over the trivial seed and recover most of the `EVIL(s)` quality: on the occupancy-based Hellinger distance every model reaches the `EVIL(s)` level on DFR, IonCh and ADP, while there is more variation between models on PFold. On the harder DFR parameter recovery, all models recover the potential strength V well, but only partially close the gap to `EVIL(s)` on the rate parameters r and b, where gpt-oss-120b comes closest. Overall the quality of the discovered algorithm is fairly robust to the choice of evolution LLM, with no single model dominating both metrics. We cannot say why the backbones differ where they do, since neither the training data nor the training procedure of these models is public; see also the limitations in Appendix A.

Table 13: Long-horizon prediction (N=5) on five real-world datasets for a longer `EVIL` run with 800 iterations. We show OTD and sMAPE$_{\Delta t}$ (both lower is better) together with a strong neural baseline (CDiff), FIM-PP variants, and the standard `EVIL` variants. Baseline results from Zeng et al. (2024); Berghaus et al. (2026). FIM-PP(zs) is FIM-PP (Berghaus et al., 2026) in zero-shot mode, FIM-PP(f) is FIM-PP after finetuning on the target dataset. `EVIL` is target-subsample evolved, `EVIL(s)` is strict zero-shot.

	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-PP(zs)	6.77	74.9	15.95	168.3	11.52	93.3	11.12	119.1	15.75	98.7
FIM-PP(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

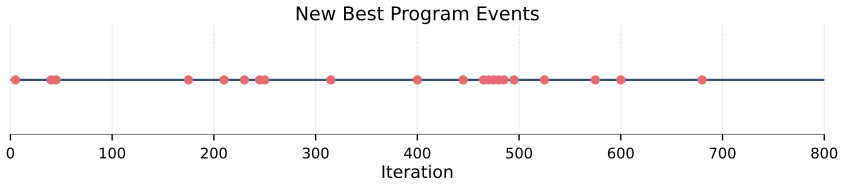


Figure 4: New best validation-set programs discovered during the 800-iteration long run. Across the run, the search found 21 new best algorithms, indicating continued progress on the validation objective even though the final gains on the benchmark test datasets remain small.

F Additional Markov Jump Process Details

F.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.

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

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

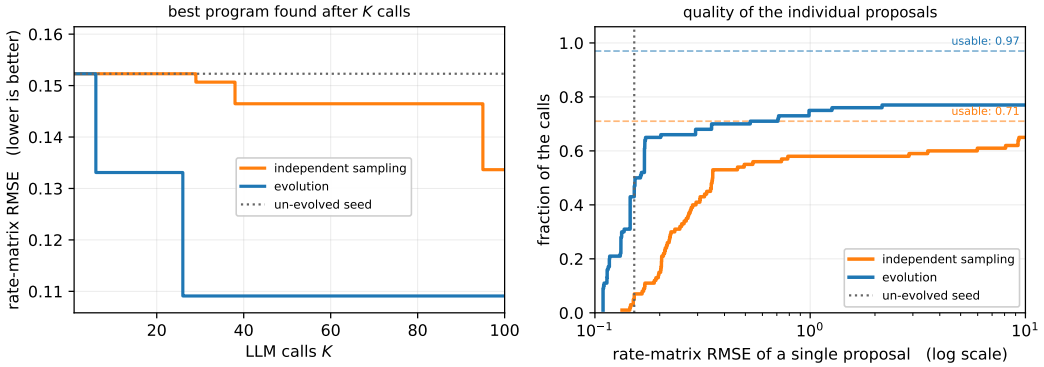


Figure 5: Evolution against independent sampling on the MJP task, at an equal budget of K LLM calls from the same seed, the same backbone and the same evaluator. **Left**: the best program available after K calls, clamped to the seed, which is what a user actually holds at budget K. **Right**: the distribution of the individual proposals, as an empirical CDF over all K calls. Normalizing by the number of calls rather than by the usable ones makes the failure rate visible as the ceiling of each curve: sampling produced no usable program on 29 of its 100 calls, so its curve stops at 0.71, while evolution reaches 0.97.

Table 14: Evolution-LLM ablation on the MJP task: time-averaged Hellinger distances ($\times 10^{-2}$, lower is better; mean (std.) over 100 histograms) for heuristics evolved for 100 iterations from the trivial seed with different LLMs. IonCh/ADP/PFold use the neural-MJP-label reference; DFR uses a regenerated clean reference dataset. The `EVIL(s)` row is reproduced from Table 17. Every row is strict zero-shot, since the MJP evolutions only ever see synthetic data.

Model	DFR	IonCh	ADP	PFold
Seed	0.35 (0.08)	0.48 (0.10)	0.92 (0.28)	0.021 (0.020)
Qwen3.5-122B-A10B	0.15 (0.03)	0.32 (0.05)	0.94 (0.24)	0.058 (0.031)
gpt-oss-120b	0.15 (0.03)	0.31 (0.05)	0.98 (0.25)	0.017 (0.016)
Gemma-4-31B-IT-MTP	0.16 (0.03)	0.30 (0.06)	0.96 (0.27)	0.016 (0.016)
`EVIL(s)`	0.16 (0.02)	0.30 (0.06)	0.93 (0.23)	0.015 (0.013)

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).

F.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 15: Evolution-LLM ablation on DFR parameter recovery: inferred rate-matrix parameters as ratios to the ground truth (1.0 is perfect), for heuristics evolved from the trivial seed with different LLMs, on the irregular-grid DFR benchmark. The `EVIL(s)` row is reproduced from Table 16. Every row is strict zero-shot.

	V	r	b
Ground Truth	1.00	1.00	1.00
Seed	1.01	0.15	0.15
Qwen3.5-122B-A10B	1.01	0.71	0.75
gpt-oss-120b	0.95	0.89	0.90
Gemma-4-31B-IT-MTP	1.01	0.76	0.80
`EVIL(s)`	0.97	1.04	1.06

F.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 F.9 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 $\mathrm{Kcv}_{\mathrm{MT325}}$, 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.

F.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 16 in Appendix F.7 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 $P = (p_1, \ldots, p_K)$ and $Q = (q_1, \ldots, q_K)$, the Hellinger distance is

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

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:

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

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.

F.5 Classical Baselines: Time Discretization and Reversibility

The classical estimators we compare against differ in two ways that matter for our benchmarks, and it is worth separating them.

The first is the treatment of time. The MSM estimators of `deeptime` (Hoffmann et al., 2022) are discrete-time methods: they fit a transition matrix at a chosen lag, and obtaining a generator from it requires a separate embedding step, in practice a matrix logarithm (Israel et al., 2001). That step is not always well defined; an empirical transition matrix need not be embeddable, in which case the logarithm produces negative off-diagonal rates that have to be clipped, or fails outright. The CT-MLE baseline, by contrast, estimates the generator directly from exposure times and transition counts (Albert, 1962), so no discrete-time approximation and no embedding are involved. `deeptime` also provides a continuous-time generator estimator, `markov.tools.estimation.rate_matrix`, which is a function on count matrices rather than one of the MSM estimator classes.

The second is reversibility. All four `rate_matrix` methods estimate a *reversible* generator, as do the default forms of the MSM estimators. Three of the four (`KL` (Kalbfleisch & Lawless, 1985), `CVE` (Crommelin & Vanden-Eijnden, 2009), and `truncated_log` (Davies, 2010)) start from the same reversible maximum-likelihood transition matrix, and on our data they agree with the reversible MSM to within 10^{-5}, which is why Table 3 collapses them into one row. Only the pseudo-generator $(T - I)/\Delta t$ behaves differently. This matters directly for the DFR: a reversible generator has no net probability current, so its entropy production is exactly zero at every potential strength, independently of how well it fits the data. The reversible estimators are therefore not merely less accurate on that benchmark, they cannot represent the quantity being measured, which is why Figure 2 does not include them. The non-reversible options are not a clean comparison either. The matrix logarithm does capture a current, but reaches the generator only through an embedding step that has no counterpart in the continuous-time estimators. The non-reversible Koopman/VAMP variant is numerically unstable

at large V, where it produces rates several times larger than the ground truth. We therefore treat the DFR entropy production as a comparison against the learned models and against our own seed, and use the rate recovery of Table 3 for the comparison against the classical estimators, where all methods are held to the same quantity.

F.6 How FIM-MJP Was Run

We use the released pretrained checkpoint (`FIM4Science/fim-mjp`, 3.5M parameters, $n_{\text{states}} = 6$) and apply it zero-shot, with no finetuning of any kind. It gets exactly the data the other rows see: one context per MJP containing all 300 observed paths, on the same normalized observation grid, with the predicted rate matrix rescaled by the same time-normalization factor used for every other method.

F.7 DFR Parameter Recovery

Table 16 reports how well each method recovers the three physical parameters of the DFR generator. `EVIL(s)` follows the overall structure well and is competitive with both learned models, with the clearest advantage on the potential strength V that controls the asymmetry of the ratchet, although FIM-MJP recovers the rate parameters r and b more accurately. The seed row shows what the three parameters are worth as a discriminator. The potential strength is easy, and even the un-evolved seed gets it to within 1%; the rate parameters are hard, and the seed underestimates them by almost an order of magnitude. As with the entropy production, we compare only against the learned models here, for the reasons given in Appendix F.5. This table also describes a single six-state system, which is why we treat the held-out rate recovery of Table 3 as the primary MJP evidence.

Table 16: 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. Closest to 1.00 per column in bold. `EVIL(s)` is strict zero-shot.

	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**
Seed (un-evolved)	1.01	0.15	0.15
`EVIL(s)`	**0.97**	1.04	1.06

F.8 Hellinger Distances on Real Data

Table 17 reports the time-averaged Hellinger distance between the observed process and processes simulated from each inferred generator, on the DFR together with the three real-world datasets. `EVIL(s)` is ahead of both learned models on DFR, IonCh and ADP, and it improves on its own un-evolved seed on DFR and IonCh, so the pretrained model does not carry here the advantage it holds at 10% label noise in Table 3. It does not, however, separate from the classical estimators: CT-MLE, the `rate_matrix` variants and the matrix logarithm all match it to within the simulation noise of the metric on every dataset, and on PFold no method is distinguishable from any other at all. We nevertheless present this table as a sanity check rather than as evidence, for two reasons. First, the metric compares state-occupancy histograms at each observation time, so it sees how often the system visits each state but not how it moves between them. A method can therefore match the occupancies well while getting the directional structure of the dynamics wrong, which is exactly the failure mode the entropy-production experiment is designed to expose. Second, and consistently with that, the metric does not separate the training-free estimators at all, as the classical rows of the table show directly: CT-MLE, the `rate_matrix` variants, the matrix logarithm and `EVIL(s)` score the same to within simulation noise, because on this much data they converge to nearly the same model. It cannot carry the MJP section on its own, which is why the primary comparison, FIM-MJP included, is the held-out rate recovery of Table 3.

Table 17: Time-averaged Hellinger distances ($\times 10^{-2}$, lower is better) between empirical and simulated processes, mean (std.) over 100 histograms. The CVE and truncated-log variants of `rate_matrix` agree with the KL variant to within the simulation noise of this metric and are not listed separately. No value is marked best: the training-free estimators agree to within that noise, and on PFold the standard deviations are as large as the means. The *Seed* row is the initial, un-evolved program used to seed the evolutionary search. `EVIL(s)` is strict zero-shot.

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)
Seed (un-evolved)	0.35 (0.08)	0.49 (0.11)	0.97 (0.31)	0.020 (0.020)
`rate_matrix` (KL)	0.14 (0.02)	0.31 (0.06)	0.97 (0.26)	0.020 (0.019)
`rate_matrix` (pseudo-generator)	0.15 (0.03)	0.31 (0.05)	0.98 (0.29)	0.020 (0.019)
Matrix logarithm	0.15 (0.03)	0.30 (0.05)	0.98 (0.29)	0.020 (0.019)
CT-MLE	0.15 (0.03)	0.31 (0.05)	0.98 (0.28)	0.020 (0.019)
`EVIL(s)`	0.16 (0.02)	0.30 (0.06)	0.93 (0.23)	0.015 (0.013)

F.9 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:

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

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

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

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

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

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):

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

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

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

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. Figure 2 should therefore be read as a derived scientific quantity: it tests whether the inferred MJP captures the correct irreversible dynamics, not only the correct state occupancies.

Figure 2 reports it as a function of the potential strength V, which is a stress test. As V grows the rates become increasingly dissimilar, until some differ by roughly four orders of magnitude and the rare transitions are only weakly represented in the observed paths. `EVIL(s)` tracks the ground truth closely over the whole range, while the un-evolved seed recovers almost none of the current, which is the clearest evidence we have that the search found something the seed did not. FIM-MJP is run zero-shot on the same simulated paths, in chunks of 300 (its pretrained context size), with the inferred generators averaged as in its own DFR experiment; of the three ways of presenting the paths to it, this is the most favourable. It overestimates the current at small V, and from $V \approx 3$ on it saturates and stops responding to the growing asymmetry altogether, so that the percentages of the true current quoted in Section 5 fall as V grows. This is a different axis of robustness from the one Table 3 measures. FIM-MJP is close to invariant under label noise, which its pretraining distribution explicitly contains; this is why it overtakes `EVIL(s)` at 10% noise. It degrades faster under a shift in the rate structure itself, which its pretraining distribution does not cover, and the strongly asymmetric DFR at large V is exactly such a shift. The `EVIL(s)` corrections, being a handful of explicit rules rather than a learned prior over generators, carry over to that regime more gracefully. The same asymmetry shows up in the rollout comparison of Appendix F.8, where `EVIL(s)` is ahead of FIM-MJP on DFR, IonCh and ADP. The two classical estimators of Table 3 that can represent a current, CT-MLE and the matrix logarithm, are drawn into the figure as well, and both saturate in the same way: at $V = 4$ they recover 52% and 70% of the true current against 86% for `EVIL(s)`. CT-MLE is the informative one, since the evolved program is built from the same statistics, and this is where its discovered corrections show their largest effect. The reversible estimators are not plotted: as explained in Appendix F.5, their entropy production is exactly zero by construction, independently of how well they fit the data, and the sweep confirms this numerically, since `rate_matrix` returns 0.00 at every V in both its KL and its pseudo-generator variant.

G Additional Imputation Dataset Details

Our imputation experiments follow the same benchmark datasets and preprocessing conventions used by the FIM imputation work (Seifner et al., 2025b). 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.

G.1 Point-Wise Missing Datasets

Eight datasets use 50% random point-wise missing patterns. Following Seifner et al. (2025b), 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.

G.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. (2025b). 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.

G.3 Synthetic Prior for EVIL(s)

The synthetic prior that EVIL(s) is evolved on is a *general* prior, specified in advance from the generative story of the FIM imputation work (Seifner et al., 2025b) rather than tuned to the benchmarks of this section. Both missingness regimes draw from the same signal distribution and use the same single noise level, so nothing in the construction is adapted per regime or per target dataset.

Signals. Each series lives on a uniform grid of 128 points on $[0, 1]$, and each of its 10 channels is drawn independently from one shared family of functions: random low-degree Chebyshev polynomials, Gaussian-process samples with an RBF, a Matérn-$3/2$ and a periodic kernel, and random low-frequency Fourier series. Roughly a third of the draws are polynomials, about half are Gaussian-process samples, and the remainder are Fourier series; the periodic kernel is what supplies seasonal structure. Every signal then receives additive Gaussian noise at a single fixed level, a few percent of its own standard deviation, and each channel is standardized to zero mean and unit variance. As in the real benchmarks, the target at a held-out position is the noisy observation rather than the clean signal, so a program cannot score by denoising.

Missingness. The signal family is drawn independently of the missingness pattern, and three synthetic datasets are built from it, each with 48 sequences. The first masks half of the positions uniformly at random, matching the point-wise benchmark rate. The second removes one contiguous window covering up to about a third of the series, and the third splits that same budget across a few windows. The first and last position are always observed, so every gap is bracketed by data. Two of the three datasets are windowed on purpose: the long-gap regime then dominates the score, and a program that only interpolates cannot win the search.

Search and selection. The search starts from the same trivial seed and uses the same minimal prompt as the target-data run. The final program is selected on a held-out synthetic split, drawn from the same generator with a different seed. No real dataset enters the search or the selection; the real benchmarks are evaluated exactly once, to produce the EVIL(s) rows of Tables 4, 5 and 18.

G.4 Classical Imputation Baselines on All Ten Dataset Variants

Table 18 reports the classical imputers on all ten dataset variants, measured by us on one CPU core under the protocol of Section 6. Our reproduction of the four TSI-Bench classical imputers agrees with the published numbers to within 0.01 on every dataset, and our cubic spline reproduces the value reported for Motion Capture in Seifner et al. (2025b) exactly, so the harness itself is validated against published classical results. The one classical number we could not reproduce is the filtered spline variant Spline(F) of Seifner et al. (2025b), reported there as 2.90 ± 0.87 and 3.00 ± 0.88 on the two Motion Capture variants. A Savitzky–Golay filter applied to a cubic spline leaves the interior of a long gap essentially unchanged, since a cubic polynomial fit reproduces the knot-free spline segment there, and we obtain 3.37 and 4.21, that is, the plain spline result. We therefore report our own measurement in Table 5 and note the discrepancy here.

The table makes the mechanism behind the evolved program visible. Retrieval and interpolation are good at opposite things. On the windowed Motion Capture benchmark, where the gaps are 16 to 24 samples long, motif retrieval is much better than linear interpolation (3.34/3.68 against 3.96/4.07) and better than the cubic spline as well. On the point-wise datasets, where at 50% missingness nearly every gap is one to three samples long, the same retrieval baseline is consistently *worse* than linear interpolation, and markedly so on GuangZhou (2.23 against 2.09) and Solar (0.520 against 0.409), because there is too little context in a short gap for a match to mean anything. A method that always retrieves therefore loses on one regime, and a method that never retrieves loses on the other. EVIL matches linear interpolation at the reported precision on all point-wise datasets but Solar, where interpolation is marginally ahead (0.409 against 0.413), and it gives the best result of all methods without per-dataset neural training on both windowed variants. What the search actually

contributed is therefore the gate: the rule that decides, from the length of the gap and the quality of the match, when not to retrieve. We regard this as the most concrete answer we can give to the question of whether the search merely rediscovered a known heuristic. Motif retrieval is certainly known (Lin, 2002), and it is the mechanism a human would reach for; the composition that makes it safe across both missingness regimes is what came out of the search.

The runtimes in Appendix H qualify the speed claim for this task in the same way. EVIL is cheaper than the cubic spline on the windowed benchmark, but 2 to 4 times more expensive than linear interpolation on the point-wise datasets, for accuracy that is identical to three decimals, because the motif search runs but rarely fires there.

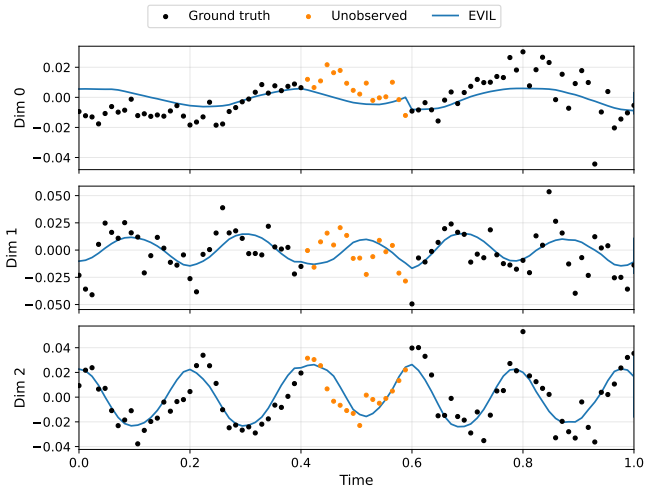


Figure 6: Imputation on Motion Capture. The motif retrieval strategy enables EVIL to predict non-trivial patterns in missing windows.

Table 18: MAE of the classical imputation baselines and the EVIL variants on all ten dataset variants (lower is better), measured by us on one CPU core. The first eight columns are 50% point-wise missingness; the last two are Motion Capture with 20% windowed missingness, reported as mean ± standard deviation over the 43 trajectories. Mean and Median are global per-channel fills, following the TSI-Bench convention (Du et al., 2024), and are not defined for the Motion Capture protocol. Best in bold. EVIL is target-subsample evolved on BEIJING, ITALY, GUANGZHOU, PEMS and the two Motion Capture variants, and strict zero-shot on the remaining four; EVIL(s) is strict zero-shot throughout.

Method	Beij.	Italy	GZ	PeMS	Ped.	Solar	ETT	Elec.	MC (PCA)	MC (No PCA)
Mean	.723	.596	6.88	.796	.738	7.81	.734	.423	–	–
Median	.684	.550	6.75	.776	.684	6.35	.706	.408	–	–
LOCF (seed)	.258	.346	3.00	.578	.356	.927	.422	.140	6.27 ± 2.64	6.38 ± 2.70
Linear interpolation	.152	.204	**2.09**	.338	.242	.409	.267	**.078**	3.96 ± 1.22	4.07 ± 1.32
Cubic spline	.276	.308	2.64	.591	.775	.431	.442	.107	3.36 ± 1.16	4.21 ± 1.42
kNN motif retrieval	.153	.204	2.23	.339	.242	.520	.268	.079	3.34 ± 1.56	3.68 ± 1.76
EVIL	.152	.204	**2.09**	.338	.242	.413	.267	**.078**	$\mathbf{2.98 \pm 2.03}$	$\mathbf{3.34 \pm 2.07}$
EVIL(s)	**.151**	**.202**	2.11	**.326**	**.238**	**.384**	**.261**	.079	3.66 ± 1.16	3.83 ± 1.22

H INFERENCE RUNTIMES

Table 19 reports the inference runtime of the final programs on all three task families, next to the neural and classical baselines. Two observations are worth making. For MJPs, the evolved program is essentially as fast as the cheapest classical estimator (0.8 ms against 0.7 ms for CT-MLE) and 3 to 6 times faster than the library MSMs, whose cost is dominated by the transition-matrix fit and the embedding step. For imputation the picture is mixed, and we state it as such. On the windowed

benchmark EVIL is cheaper than the cubic spline, but on the point-wise datasets it is 2 to 4 times more expensive than linear interpolation (1797 ms against 560 ms for the whole ELECTRICITY dataset) at the same accuracy. The reason is that its motif search runs on every gap but almost never fires when the gaps are short.

Table 19: Inference runtime of the final programs, on one Apple M3 CPU core unless stated otherwise. TPP: one full $N{=}5$ evaluation. MJP: one MJP inferred from 300 observed paths. Imputation: one Motion Capture sequence in the 50-dimensional space (no PCA). All values exclude data loading and the one-off cost of the search (15 to 40 minutes and under $1 per task). The neural imputation baselines are not timed, since all but FIM-ℓ require per-dataset training and FIM-ℓ is a large pretrained transformer.

Task	Method	Time
TPP	FIM-PP(zs) (neural, A100 40GB)	964 s
	Poisson / renewal / Markov-renewal	0.8 s
	Self-correcting	0.9 s
	ACD	1.8 s
	EVIL	5 s
MJP	FIM-MJP (neural)	71 ms
	Seed (un-evolved)	27.9 ms
	Reversible MSM (`deeptime`)	4.7 ms
	Matrix-log MSM (non-reversible)	2.3 ms
	CT-MLE	0.7 ms
	EVIL(s)	0.8 ms
Imputation	Cubic spline	13.8 ms
	kNN motif retrieval	0.44 ms
	Linear interpolation	0.35 ms
	LOCF (seed)	0.35 ms
	EVIL	10.6 ms

I APPLYING EVIL TO A NEW PROBLEM

Since the search machinery is generic, most of the work in applying EVIL to a new inference problem lies in four artifacts that the user provides. We describe them in the order in which we found it useful to write them.

1. The interface. Fix the signature of the function to be discovered, and make it operate on one problem instance at a time: context observations in, prediction out, plain NumPy arrays on both sides. Our three interfaces are `predict_next_events(target histories, context sequences, num_classes)`, `estimate_mjp_parameters(observation grid, observed states, sequence lengths, n_states)`, and `impute(values with NaNs, times, prediction mask)`. Two properties are worth designing for. The interface should be dimension-agnostic wherever the problem allows it: this is what makes the resulting program transfer to datasets with more marks or more states than were seen during the search. It should also contain no slot for fitted parameters, which is what keeps the result zero-shot at test time.

2. A trivial seed program. Write the most boring correct implementation of the interface: the global median gap and the majority mark for point processes, normalized transition counts for MJPs, last observation carried forward for imputation. Keeping the seed trivial matters more than it may appear. A strong seed makes it hard to tell afterwards whether the search contributed anything, and in our experience it also anchors the LLM to the structure it already sees, so it proposes refinements rather than alternatives.

3. A scoring function. The evaluator must return a single scalar (we return additional diagnostic metrics alongside it, which are logged but not optimized), must be cheap enough to run a few hundred times, and must be computed on data that comes from the training split only. Two practical points. Impose a hard timeout, both because a proposal can fail to terminate and because a proposal that is accurate but ten times slower is usually not what is wanted; we used 60 s per evaluation for point processes and MJPs. Also check the dynamic range of the score: if one term dominates the sum, the

search is effectively optimizing that term alone. In our MJP score, for example, the cross-entropy term contributes most of the magnitude but almost none of the variation between programs, since any program that counts first observations lands within 0.014 of the entropy floor, so the rate RMSE is what actually drives selection.

4. A problem description for the prompt. A short system prompt stating the task, the constraints (in our case: NumPy only, no deep learning frameworks, all code inside the evolve block), and the domain facts the LLM cannot infer from the interface alone. For MJPs, for instance, it matters that the observations are discrete snapshots rather than exact jump times, and that they are noisy; without those two sentences the proposals treat the data as a fully observed chain. We deliberately kept the prompts generic and did not tune them against benchmark results. Appendix K gives our prompts verbatim.

Running the search and selecting the result. With those four artifacts in place, the search is run for a fixed budget: 100 iterations throughout this paper, which took 14 to 37 minutes and under $1 of API cost per task. The final program is the one with the best score on the same data that the search scored on. Nothing else is selected, and in particular no test score is consulted. Two things are worth watching. Longer runs produce longer programs without necessarily producing better ones (Appendix E), so a budget in the low hundreds is a reasonable default. And because the score is computed on a subsample, a program can win on the score while being no better in general; the cheapest guard we found is to re-evaluate the top few programs on a second, freshly drawn sample before accepting one.

J 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

```
"""
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(s)` 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(s)` 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(s)` 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(s)` 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)
```

```python


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
```

Pseudocode 5: `EVIL(s)` heuristic for Time Series Imputation

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

1. If the series is entirely missing, leave it for the cross-channel fallback.
2. Fill every missing position by time-aware linear interpolation as a baseline.
3. Detect contiguous blocks of missing values.
4. For each block longer than one sample:
   (a) Find the nearest observed point on either side of the gap; if the gap is not bracketed on both sides, keep the linear result.
   (b) Collect a window of observed points immediately before and after the gap.
   (c) Fit a monotone (shape-preserving) spline to those points and evaluate it across the gap, which avoids the overshoot of an unconstrained cubic fit.
   (d) Clip the result to the observed range of the window.
5. Fill any channel that is still entirely missing with the mean over the remaining channels at each timestamp, falling back to the global mean.

Algorithm 5: `EVIL(s)` algorithm for time series imputation

```python
"""
Time series imputation heuristic (evolvable block), synthetic-prior variant.

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 scipy.interpolate import PchipInterpolator

MIN_GAP_LEN = 2
WINDOW = 7

def _impute_1d(out: np.ndarray, times: np.ndarray) -> None:
    """
    In-place imputation for a single 1D series.
    Short gaps are left to linear interpolation; longer gaps get a local monotone
    spline fitted to a window of observed points on either side.
    """
    valid = np.isfinite(out)

    # Nothing observed: keep the NaNs so the caller can fill the whole channel.
    if not np.any(valid):
        out[:] = np.nan
        return

    if np.all(valid):
        return

    # Linear interpolation as the baseline for the whole series.
    out[:] = np.interp(times, times[valid], out[valid])

    # Split the missing positions into contiguous blocks.
    nan_idx = np.where(~valid)[0]
    if nan_idx.size == 0:
        return
    gap_ends = np.where(np.diff(nan_idx) > 1)[0]
    start_idxs = np.concatenate(([nan_idx[0]], nan_idx[gap_ends + 1]))
    end_idxs = np.concatenate((nan_idx[gap_ends], [nan_idx[-1]]))

    for s, e in zip(start_idxs, end_idxs):
        # Short gap: keep the linear result.
        if e - s + 1 < MIN_GAP_LEN:
            continue

        # Nearest observed point on either side of the gap.
```

```
        left = s - 1
        while left >= 0 and not np.isfinite(out[left]):
            left -= 1
        right = e + 1
        while right < out.shape[0] and not
        ↪  np.isfinite(out[right]):
            right += 1

        # One side missing: keep the linear result.
        if left < 0 or right >= out.shape[0]:
            continue

        # Window of observed points bracketing the gap.
        left_idxs = np.arange(max(0, left - WINDOW + 1), left
        ↪  + 1)
        right_idxs = np.arange(right, min(out.shape[0], right
        ↪  + WINDOW))
        sel = np.concatenate((left_idxs, right_idxs))
        sel = sel[np.isfinite(out[sel])]

        # A shape-preserving PCHIP spline avoids the
        ↪  overshoot of a cubic fit,
        # and clipping to the window range is a further
        ↪  safeguard.
        if sel.size >= 4:
            pchip = PchipInterpolator(times[sel], out[sel],
            ↪  extrapolate=True)
            vals = pchip(times[s:e + 1])
            out[s:e + 1] = np.clip(vals, out[sel].min(),
            ↪  out[sel].max())

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 positions.
    observation_times: (T,) timestamps for each time step.
    prediction_mask: (T, D) True at positions we must
    ↪  predict.
    """
    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)
```

```
    # A channel that is still entirely missing is filled with the mean over the
    # other channels at each timestamp, falling back to the global mean.
    overall_mean = np.nanmean(observation_values)
    if np.isnan(overall_mean):
        overall_mean = 0.0

    all_nan_cols = ~np.isfinite(out).any(axis=0)
    if np.any(all_nan_cols):
        temporal_mean = np.nanmean(out[:, ~all_nan_cols], axis=1, keepdims=True)
        if np.isnan(temporal_mean).all():
            out[:, all_nan_cols] = overall_mean
        else:
            out[:, all_nan_cols] = temporal_mean
    out[~np.isfinite(out)] = overall_mean

    return out
```

K 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 search scored on BEIJING, ITALY, GUANGZHOU, PEMS, MOTIONCAPTURE_PCA and MOTIONCAPTURE_NOPCA; PEDESTRIAN, SOLAR, ETT_H1 and ELECTRICITY were held out from the search entirely and are only ever evaluated on (Appendix B). 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.
```