RELOBGEN: REPLAYABLE LIMIT ORDER BOOK MESSAGE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose ReLOBGen, a method for generating limit order book (LOB) messages that are replayable by construction. Replayability is required for closed-loop market simulation, yet existing LOB message generators may produce non-replayable raw messages, i.e., messages inconsistent with the current market state. These generators therefore rely on post-hoc correction or rejection followed by resampling, which may alter the replayed message distribution or increase inference cost. ReLOBGen instead ensures replayability during generation: it selects the referenced order from the resting orders in the current LOB and then generates the remaining message fields to be consistent with that order and the market state. For realistic reference selection, ReLOBGen samples from a learned distribution over eligible resting orders, efficiently computed from cached order representations and a context-dependent query. It then enforces the consistency of the remaining fields by masking out invalid tokens. Together, these components enable efficient generation of realistic messages without post-hoc correction or resampling. In 500-message rollouts, ReLOBGen achieves 100% replayability, improves market realism, particularly for top-of-book statistics and the relative prices of LOB messages, and provides a 2.7–3.6$\times$ speedup per replayed message over the LOBS5 baseline.

1 INTRODUCTION

A limit order book (LOB) contains buy and sell orders waiting to be traded, called resting orders. Each resting order is characterized by its price, size, and timestamp. LOB messages specify changes to the book: add events introduce new resting orders, while cancellation and execution events reduce or remove a referenced resting order. Recent work (Nagy et al., 2023; Wheeler & Varner, 2024) models sequences of LOB messages autoregressively, learning to generate the next message from preceding messages and LOB states. These models have a range of potential applications in financial markets, such as training and evaluating trading strategies and analyzing market impact (Nagy et al., 2023; Li et al., 2025). To simulate a market for these applications, the model and a rule-based LOB simulator (Byrd et al., 2020; Frey et al., 2023) operate in a closed loop, as illustrated in Figure 1. At each step, the model generates a message, and the simulator updates the LOB according to that message; we refer to this update as *replay*. The updated LOB then conditions the next generation step.

However, an LOB message generator may produce raw messages that cannot be replayed against the current market state, such as a cancellation message that references a nonexistent resting order or whose size exceeds the referenced order's remaining size. For example, over 25% of raw generation attempts in our reproduction of the LOBS5 baseline are not replayable as generated. Existing approaches therefore rely on method-specific post-processing to obtain a replayable message from a raw output $\widehat{M}_t$. This includes correction or rejection followed by resampling if correction fails, as illustrated in the post-processing stage of Figure 1. These post-processing procedures are often heuristic: LOBS5 (Nagy et al., 2023) and MarketGPT (Wheeler & Varner, 2024) apply different fallback matching rules to identify the order referenced by a cancellation message, whereas MarS (Li et al., 2025) projects the generated cancellation price to the nearest available resting-order price. LOBS5 and MarketGPT also interpret the generated cancellation quantity differently: LOBS5 treats it as an absolute amount, whereas MarketGPT preserves the generated cancellation

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.

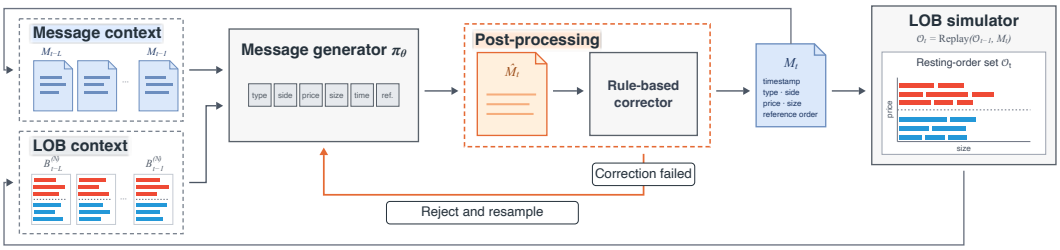


Figure 1: **Market simulation with an LOB message generator.** At each step, the generator produces a raw message $\widehat{M}_t$. When $\widehat{M}_t$ is not replayable, existing approaches correct it when possible or otherwise reject it and resample. ReLOBGen, however, generates replayable messages by construction and requires no post-processing. The LOB simulator replays the resulting message M_t to update the market state, which conditions the next generation step.

ratio. Such corrections may alter message semantics and shift the distribution of replayed messages, while resampling increases inference cost. This problem is structural: replayability depends on the exact current market state, which a finite message context and aggregated LOB states may not fully specify (see Appendix D).

We introduce ReLOBGen, which generates replayable LOB messages by construction and thus requires neither post-hoc correction nor resampling. Its design satisfies two jointly sufficient conditions for replayability: reference eligibility and event-order compatibility—a non-add event must reference an eligible resting order, and the event order must be consistent with the event, the market state, and any reference order it is applied to. ReLOBGen's message representation encodes the reference order using the target resting order's current attributes, such as price and remaining quantity, so that these attributes can constrain the event-order fields. During generation, ReLOBGen first generates the event and then selects the reference order from the resting orders eligible for that event, ensuring reference eligibility. Finally, it generates the event-order fields while masking out invalid token values, ensuring event-order compatibility. For realistic reference selection, ReLOBGen samples from a learned distribution over eligible resting orders, efficiently computed by dot-product scoring of cached order representations against a context-dependent query.

Empirically, ReLOBGen achieves 100% replayability, improves rollout realism, and generates messages 2.7–3.6× faster than the LOBS5 baseline (Nagy et al., 2023) in 500-message rollouts on GOOG and INTC. Using the same S5 (Smith et al., 2023) backbone as the baseline, ReLOBGen achieves this speedup through fewer autoregressive decoding steps and the elimination of resampling. In terms of rollout realism, ReLOBGen more closely matches the marginal event-type distribution and shows broad improvements across unconditional, conditional, and market-impact evaluations with LOB-Bench (Nagy et al., 2025). These improvements are particularly pronounced in metrics capturing top-of-book statistics and the prices of submitted and canceled orders relative to the current book. An ablation shows that learned reference selection improves rollout realism over uniform selection from the same eligible resting orders.

Overall, ReLOBGen makes every generated message replayable by construction—selecting reference orders from the eligible resting orders and using the selected orders to constrain event-order generation. Our results show that this guarantee can be achieved alongside improved rollout realism and computational efficiency. Generated messages are replayed without modification, preserving their semantics and avoiding distribution changes introduced by method-specific correction rules. Together, these results establish ReLOBGen as a practical method for efficiently generating replayable and realistic LOB messages.

2 Related Work

2.1 Modeling Financial Market Microstructure

Data-driven modeling of financial markets spans multiple targets, from future price movements to the evolution of the limit order book. DeepLOB (Zhang et al., 2019) predicts future mid-price

movements from recent LOB observations. Because the mid-price provides only a compressed representation of market state, subsequent work directly forecasts or generates the joint evolution of prices and queued volumes across multiple book levels (Jung & Lee, 2025; Backhouse et al., 2025).

Another line of work models order flow—the sequence of events that update the book—using stochastic processes. Smith et al. (2003) treat order flow as independent random events and analyze continuous double-auction statistics, while Cont et al. (2010) model limit, market, and cancellation events as Poisson flows calibrated to high-frequency data. Huang et al. (2015) make event intensities depend on the current book state in a queue-reactive model, whereas Bacry et al. (2015) describe multivariate Hawkes processes that capture temporal clustering and self- and cross-excitation through history-dependent intensities.

More recent work uses neural autoregressive models to generate order flow. LOBS5 (Nagy et al., 2023) uses an S5-based state-space model (Smith et al., 2023) to generate tokenized LOB messages from LOBSTER data (Huang & Polak, 2011) for two Nasdaq equities, conditioned on past messages and LOB states. MarketGPT (Wheeler & Varner, 2024) uses a decoder-only Transformer (Vaswani et al., 2017) to generate tokenized ITCH messages from message histories, with full-depth Nasdaq pretraining across 20 equities and asset-specific fine-tuning. MarS (Li et al., 2025) combines order-level and order-batch models to generate orders conditioned on past orders and LOB states, training on 500 liquid Chinese equities. TradeFM (Kawawa-Beaudan et al., 2026) scales Transformer-based order-flow generation to over 9,000 US equities, conditioning on LOB message streams.

2.2 Evaluating message-generation models

Message-generation models are commonly evaluated using next-token perplexity and market realism in autoregressive rollouts (Nagy et al., 2023; Kawawa-Beaudan et al., 2026). LOB-Bench (Nagy et al., 2025) assesses realism by comparing distributions of market microstructure statistics, such as spreads and message inter-arrival times, extracted from real and generated messages and LOB trajectories. The 21 metrics span six groups: top-of-book conditions (*State*), event timing (*Times*), resting liquidity (*Volumes*), the locations of order submissions and cancellations, measured by price distance and book-level rank (*Depths* and *Levels*, respectively), and trading activity and order-flow imbalance (*Trades*). These comparisons cover unconditional and conditional distributions, as well as error accumulation measured by changes in distributional divergence over the rollout horizon. It also evaluates market impact through price responses to order events and temporal correlations between order events. Additionally, we evaluate raw-message replayability by measuring how often generated messages require correction or rejection and whether they violate reference eligibility or event-order compatibility.

3 Problem formulation

3.1 LOB message generation model

We first define the main objects used in LOB message generation. Let $\mathcal{O}_{t-1}$ denote the set of resting orders at step $t-1$. An LOB message $M_t = (E_t, R_t, X_t)$ describes a change to this set: E_t specifies the event type and side (bid or ask), R_t denotes the reference order, and X_t denotes the event order. We consider four event types: add inserts a new resting order, cancel partially reduces a referenced order's remaining quantity, delete removes the referenced order entirely, and execute reduces a referenced order's remaining quantity through a trade. The orders involved in these events are represented by X_t and R_t, each containing price, size, and timestamp fields. For add events, X_t describes the newly submitted order, and no reference order is required. For non-add events, R_t identifies the referenced resting order, and X_t specifies the price and quantity of the operation on that order. Replaying M_t updates the resting-order set to $\mathcal{O}_t$. Aggregating the remaining quantities in $\mathcal{O}_t$ by side and price and retaining the best N occupied price levels on each side yields the N-level LOB state $B_t^{(N)}$, which does not retain individual resting orders.

An LOB message generation model π_θ learns the conditional distribution of the next message M_t given the preceding L LOB messages $M_{t-L:t-1}$ and N-level LOB states $B_{t-L:t-1}^{(N)}$. Each message is tokenized and modeled autoregressively using architectures such as Transformers (Vaswani et al., 2017) or state-space models (Smith et al., 2023). The generation model used in this work follows

Table 1: **Event-specific replayability conditions.** Here, $\text{side} = E_t[\text{side}]$, $\mathcal{O}_{t-1}^{(\text{side})}$ contains the resting orders on that side, and R_t carries the referenced order's current attributes. BestAsk and BestBid denote the lowest ask and highest bid prices in the current LOB state. FrontOfQueue returns the oldest order resting at the best price on that side. Under the LOB simulator's price–time priority, an execute event allows only this order as its reference.

Event type	Eligible resting-order set $\mathcal{C}_t$	Event-order constraints
Add	–	Bid: $X_t[\text{price}] < \text{BestAsk}$; Ask: $X_t[\text{price}] > \text{BestBid}$.
Cancel	$\{o \in \mathcal{O}_{t-1}^{(\text{side})} : o[\text{size}] > 1\}$	$X_t[\text{price}] = R_t[\text{price}]$ and $1 \le X_t[\text{size}] < R_t[\text{size}]$.
Delete	$\{o \in \mathcal{O}_{t-1}^{(\text{side})} : o[\text{size}] > 0\}$	$X_t[\text{price}] = R_t[\text{price}]$ and $X_t[\text{size}] = R_t[\text{size}]$.
Execute	$\{\text{FrontOfQueue}(\mathcal{O}_{t-1}^{(\text{side})})\}$	$X_t[\text{price}] = R_t[\text{price}]$ and $1 \le X_t[\text{size}] \le R_t[\text{size}]$.

LOBS5 (Nagy et al., 2023), using an S5 architecture and representing each message M_t as a sequence of 22 field-wise tokens $z_{t,1:22}$ (see Table 8). The model is trained to minimize the standard autoregressive negative log-likelihood:

$$\mathcal{L}_{\text{NLL}}(\theta) = -\mathbb{E}_{t,k}\left[\log \pi_\theta\left(z_{t,k} \mid M_{t-L:t-1}, B^{(N)}_{t-L:t-1}, z_{t,<k}\right)\right]. \tag{1}$$

3.2 Market simulation and replayability

To simulate a market trajectory autoregressively, the message generation model π_θ and the LOB simulator alternate between message generation and replay, as illustrated in Figure 1. At each step, the model generates a message conditioned on the preceding message and LOB state histories. The simulator replays the message M_t against the current resting-order set:

$$\mathcal{O}_t = \text{Replay}(\mathcal{O}_{t-1}, M_t), \tag{2}$$

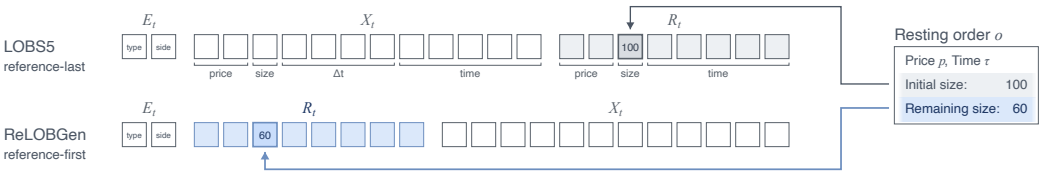
where Replay denotes the event-specific deterministic state-transition operator used by the LOB simulator. The message M_t and the LOB state $B_t^{(N)}$ aggregated from $\mathcal{O}_t$ are appended to their respective histories to condition the next generation step.

However, a message $\widehat{M}_t$ generated by π_θ may not be replayable against the current resting-order set $\mathcal{O}_{t-1}$. For example, a non-add event may reference an order that is absent from $\mathcal{O}_{t-1}$ or ineligible for that event. Even with an eligible reference, the event order may violate event-specific constraints, such as exceeding the reference order's remaining quantity; an add event may instead violate the price constraint imposed by the current market state. In our reproduction of LOBS5 (ref.-last), over 25% of raw generation attempts require correction or rejection on both stocks (Table 2). The larger MarketGPT model also reports that approximately 7% of generated messages cannot be corrected and require resampling (Wheeler & Varner, 2024). Based on the LOB simulator's replay rules, we define replayability as follows.

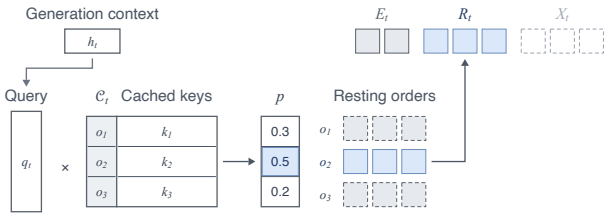
Definition 1 (Replayability)**.** A message $M_t = (E_t, R_t, X_t)$ is replayable with respect to the current resting-order set $\mathcal{O}_{t-1}$ if it satisfies both conditions:

- **Reference eligibility.** For a non-add event, the reference order R_t must belong to the eligible resting-order set $\mathcal{C}_t(E_t, \mathcal{O}_{t-1})$ defined in Table 1.
- **Event-order compatibility.** The event order X_t satisfies the constraints in Table 1.

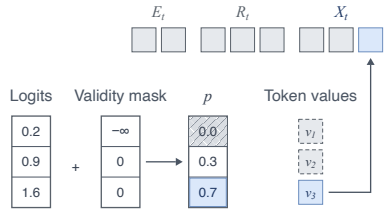
When a generated message $\widehat{M}_t$ is not replayable, existing approaches use method-specific post-processing to obtain a replayable message M_t (Nagy et al., 2023; Wheeler & Varner, 2024; Li et al., 2025). This includes correction or rejection followed by resampling, with event-specific procedures detailed in Appendix A. Although this post-processing makes M_t replayable, it introduces two limitations. First, method-specific corrections may modify generated fields or reinterpret their semantics, shifting the distribution of replayed messages away from that of raw model outputs. Second, resampling rejected messages increases inference cost. These limitations motivate generating messages that are replayable by construction while preserving rollout realism.



(a) Reference-first message representation



(b) Resting-order selection



(c) Inference-time masking

Figure 2: **ReLOBGen: mechanisms for replayability by construction.** (a) The message representation places R_t before X_t and encodes the referenced order's remaining size rather than its initial submitted size. (b) ReLOBGen samples R_t from a learned distribution over the eligible resting-order set $\mathcal{C}_t$. This distribution is efficiently computed by applying a softmax to the scaled dot products between a query derived from the generation context (the hidden state after processing E_t) and the cached order keys. (c) ReLOBGen masks invalid token values and samples from the remaining valid support during X_t generation.

4 ReLOBGen: Replayable LOB message generation

ReLOBGen generates each LOB message by first generating the event E_t, then resolving the reference order R_t, and finally generating the event order X_t. For non-add events, resolving the referenced resting order first allows the event order's price and quantity to be generated within the constraints imposed by that order, which makes reference-first generation a natural choice. We adapt the message representation to support this reference-first generation process (Section 4.1). During generation, ReLOBGen first generates the event E_t. For non-add events, ReLOBGen then selects the reference order from the eligible resting-order set (Section 4.2). Finally, it generates the event order to be consistent with the event, any selected reference order, and the current market state by masking out invalid token values (Section 4.3). Together, these mechanisms produce replayable messages by construction without post-hoc correction or resampling.

4.1 Reference-first message representation

For ReLOBGen, we make two modifications to the LOBS5 (Nagy et al., 2023) message representation, as illustrated in Figure 2 (a), while retaining its field-wise tokenization and vocabulary (see Table 8 in Appendix B.1). First, we place R_t before X_t in the token sequence, yielding the order $E_t \rightarrow R_t \rightarrow X_t$ and making the reference available to determine the constraints on X_t before its generation. Second, we represent R_t using the referenced resting order's current attributes so that these constraints reflect the current market state. Specifically, we set $R_t[\text{size}]$ to the order's remaining size in $\mathcal{O}_{t-1}$ rather than the initial submitted size used by LOBS5. We train the message generation model π_θ on this representation using the negative log-likelihood objective in Section 3.1.

4.2 Resting-order selection for reference eligibility

Resting-order selection. ReLOBGen guarantees the reference eligibility condition in Definition 1 by selecting R_t only from the eligible resting-order set $\mathcal{C}_t(E_t, \mathcal{O}_{t-1})$ specified in Table 1. Note that the number of model forward passes per message is reduced by selecting R_t rather than autoregressively generating its eight tokens, and non-add events with no eligible resting orders are masked during event generation.

While selecting any eligible resting order guarantees reference eligibility, the particular choice still affects the realism of the generated LOB messages. ReLOBGen therefore samples R_t from a learned categorical distribution $p_\phi(\cdot \mid \mathbf{h}_t, \mathcal{C}_t)$ over the eligible resting orders, where $\mathbf{h}_t$ denotes the generation context up to and including E_t. With the base model frozen, we train this distribution by minimizing the cross-entropy for the observed reference order $R_t^\star$:

$$\mathcal{L}_{\text{sel}} = -\mathbb{E}_t \left[\log p_\phi(R_t^\star \mid \mathbf{h}_t, \mathcal{C}_t) \right]. \tag{3}$$

Efficient resting-order selection with cached keys. ReLOBGen implements resting-order selection efficiently by exploiting a structural property of LOB messages: each message affects only one resting-order entry. We therefore cache a key $\mathbf{k}_o$ for each resting order and reuse the keys of unchanged orders. Using query–key scoring similar to dense retrieval (Karpukhin et al., 2020), each eligible resting order $\mathbf{o} \in \mathcal{C}_t(E_t, \mathcal{O}_{t-1})$ is scored by the scaled dot product between its cached key $\mathbf{k}_o$ and a query $\mathbf{q}_t$ representing the current generation context, as illustrated in Figure 2 (b):

$$s_\phi(\mathbf{o}, \mathbf{h}_t) = \mathbf{q}_t^\top \mathbf{k}_o / \sqrt{d}, \tag{4}$$

where d is the key and query dimension. Applying a softmax over these scores within the eligible resting-order set yields $p_\phi(\cdot \mid \mathbf{h}_t, \mathcal{C}_t)$.

To keep cached keys reusable as the market moves, we encode resting orders relative to a fixed mid-price anchor rather than the current mid-price, while the query incorporates the current generation context and the displacement of the current mid-price from this anchor. We represent generation context $\mathbf{h}_t$ using the base model's hidden state and compute the order key and shared query using lightweight MLPs, f_{order} and f_{query}:

$$\mathbf{k}_o = f_{\text{order}}(\mathbf{o}; \text{mid}_{\text{anchor}}), \qquad \mathbf{q}_t = f_{\text{query}}(\mathbf{h}_t, \text{mid}_{t-1}; \text{mid}_{\text{anchor}}). \tag{5}$$

Here, $\text{mid}_{\text{anchor}}$ is the fixed mid-price anchor, and mid_{t-1} is the mid-price at step $t-1$.

At the start of each rollout, we set $\text{mid}_{\text{anchor}}$ to the initial mid-price. When an order is added or updated, we compute its key from the order's current attributes; when it is removed, we discard its key; we maintain cached keys for unaffected orders. When training f_{order} and f_{query}, we sample $\text{mid}_{\text{anchor}}$ from earlier mid-prices in the context to expose them to price drift relative to the anchor.

4.3 Inference-time masking for event-order compatibility

ReLOBGen guarantees event-order compatibility by restricting each field in X_t to the valid support determined by E_t, R_t, and the current market state, as specified in Table 1. Figure 2 (c) illustrates this process: the model masks invalid tokens and samples from the remaining support, as in constrained decoding for structured generation (Scholak et al., 2021). When the valid support is a singleton, ReLOBGen directly assigns the unique valid token, further reducing the number of autoregressive model forward passes. The pretrained model assigns negligible probability mass to invalid tokens on teacher-forced test windows—less than 0.14% on GOOG and 0.29% on INTC, suggesting a limited impact of forcing and masking.

5 Experiments

5.1 Experimental setup

Data and model training. We use market-by-order (MBO) data for GOOG and INTC from Databento[1], with 2025 data for training and separate periods in January 2026 for validation and testing. We follow the preprocessing and field-wise tokenization of LOBS5 (Nagy et al., 2023), except for the reference-field semantics described in Section 4.1 (see Appendix B.1 for details). We train 35M-parameter models using the same S5 backbone as the scaled-up LOBS5 in Nagy et al. (2025), with reference-last and reference-first token orders. We additionally train two lightweight MLP projection heads with 2.3M parameters in total for learned reference selection (see Appendix B.2 for training details).

[1] https://databento.com/portal/catalog/us-equities#XNAS.ITCH

Table 2: **Post-processing rates, replayability violations, and runtime.** Runtime is measured on an NVIDIA RTX A6000 GPU and reported as mean ± standard deviation. Aborted rollouts—59 and 17 for LOBS5 (ref.-last) and LOBS5 (ref.-first) on GOOG, respectively, and two for each baseline on INTC—are excluded from evaluation. Neither ReLOBGen nor Ref.-first + uniform requires any restarts on either stock. Lower is better, and bold indicates the lowest value.

		Post-processing		Replayability violations		Runtime (ms)	
Ticker	Method	Correction	Rejection	Reference	Event-order	Per replayed msg.	Per attempt
GOOG	LOBS5 (ref.-last)	13.5%	25.7%	39.1%	0.4%	602 ± 164	447 ± 119
	LOBS5 (ref.-first)	11.7%	27.6%	39.5%	0.1%	591 ± 145	426 ± 100
	ReLOBGen	**0.0%**	**0.0%**	**0.0%**	**0.0%**	$\mathbf{227 \pm 12}$	$\mathbf{227 \pm 12}$
INTC	LOBS5 (ref.-last)	9.7%	16.3%	25.6%	0.5%	1216 ± 726	1018 ± 606
	LOBS5 (ref.-first)	11.5%	13.2%	24.5%	0.2%	1058 ± 850	921 ± 744
	ReLOBGen	**0.0%**	**0.0%**	**0.0%**	**0.0%**	$\mathbf{341 \pm 150}$	$\mathbf{341 \pm 150}$

Rollout evaluation. We compare the following configurations on the same 1,000 rollout windows per stock, with 500 successfully replayed messages per rollout, matching the rollout length reported in LOB-Bench (Nagy et al., 2025) (see Appendix B.3 for details). ReLOBGen combines the reference-first model with learned selection from the eligible resting-order set and inference-time masking. Our baselines autoregressively generate messages with either model and apply the LOBS5 post-processing procedure (see Appendix A.1 for details); we denote them by LOBS5 (ref.-last) and LOBS5 (ref.-first). Ref.-first + uniform uses the reference-first model and uniform reference selection with inference-time masking. All configurations start from the same exact resting-order set, which is maintained by the JAX-LOB simulator (Frey et al., 2023) and available to both ReLOBGen and baseline post-processing. We assess replayability on raw generation attempts before post-processing and evaluate rollout realism from replayed messages and resulting LOB trajectories using LOB-Bench and the marginal event-type distribution.

5.2 Replayability

Table 2 summarizes post-processing and replayability violation rates, with an event-wise breakdown in Appendix C.1. For non-add events, event-order compatibility is assessed against the generated reference fields, regardless of whether the reference is eligible. Both LOBS5 baselines frequently require correction or rejection, with reference eligibility violations far more frequent than event-order compatibility violations. This gap may partly reflect the difficulty of identifying eligible resting orders, many of which are not individually specified by the finite message context or the aggregated LOB state, as analyzed in Appendix D. Both baselines also require rollout restarts after 100 consecutive failed message-generation attempts. On the other hand, ReLOBGen satisfies both replayability conditions by construction, recording zero violations and requiring no post-processing throughout the evaluated rollouts. Note that LOBS5 (ref.-last) still requires post-processing when its resting-order set is initialized with INIT orders, as shown in Appendix C.2.

Relative to LOBS5 (ref.-last), ReLOBGen achieves $2.7\times$ and $3.6\times$ speedups in runtime per replayed message on GOOG and INTC, respectively. Runtime covers the full closed-loop simulation, including message generation, post-processing when required, and simulator state updates. These gains come from reduced autoregressive decoding and the elimination of post-processing. Selecting the reference order rather than generating its tokens autoregressively and forcing singleton-valid field tokens reduces non-add decoding from 17 to 7–8 model forward passes. Eliminating rejection also removes the $1/(1-r)$ factor in attempts per replayed message, where r denotes the rejection rate. The larger speedup on INTC might reflect higher reference-matching costs in the baseline due to its larger resting-order population.

5.3 Rollout realism

Beyond generating fully replayable messages, ReLOBGen also improves rollout realism over the LOBS5 baseline, with broadly lower unconditional L_1 distances across metric groups, as shown in Table 3. These improvements are particularly pronounced in the State, Depths, and Levels metric

Table 3: **LOB-Bench evaluation: metric-group summary of unconditional L_1 distances.** Overall averages all metrics, while each metric-group column averages the metrics in its group. Lower is better, and bold indicates the lowest value.

			Metric group					
Ticker	Method	Overall	State	Times	Volumes	Depths	Levels	Trades
GOOG	LOBS5 (ref.-last)	0.27	0.35	0.16	0.25	0.40	0.21	0.23
	Ref.-first + uniform	0.24	0.31	0.20	0.22	0.23	0.20	0.27
	ReLOBGen	**0.17**	**0.24**	**0.15**	**0.18**	**0.20**	**0.11**	**0.18**
INTC	LOBS5 (ref.-last)	0.22	0.14	**0.14**	0.20	0.32	0.17	0.27
	Ref.-first + uniform	0.30	0.10	0.35	0.27	0.36	0.34	0.32
	ReLOBGen	**0.11**	**0.05**	0.21	**0.16**	**0.07**	**0.04**	**0.15**

Table 4: **LOB-Bench evaluation: conditional L_1 distances and market-impact discrepancies.** Market-impact discrepancies cover market orders (MO), limit orders (LO), and cancellations (CA), with subscripts 0 and 1 for events without and with an immediate mid-price change. Conditional 99% percentile bootstrap confidence intervals have half-widths below 0.004. Lower is better, and bold indicates the lowest value.

		Conditional (L_1)			Market impact					
Ticker	Method	ask vol. | spread	spread | time	spread | volatility	MO_0	MO_1	LO_0	LO_1	CA_0	CA_1
GOOG	LOBS5 (ref.-last)	**0.14**	0.63	0.46	23.4	16.9	5.5	8.7	4.5	**4.7**
	Ref.-first + uniform	0.16	0.51	0.57	12.1	13.9	10.5	4.9	7.6	8.8
	ReLOBGen	0.18	**0.37**	**0.42**	**7.2**	**10.4**	**1.7**	**3.1**	**3.3**	5.3
INTC	LOBS5 (ref.-last)	0.22	0.22	0.16	9.9	5.9	2.3	3.5	**0.4**	11.8
	Ref.-first + uniform	0.49	0.15	0.09	7.2	**5.4**	**0.8**	**2.2**	1.3	11.6
	ReLOBGen	**0.20**	**0.05**	**0.06**	**6.2**	6.3	**0.8**	**2.2**	0.5	**11.2**

groups. State captures top-of-book statistics, while Depths and Levels characterize the locations of order submissions and cancellations, measured by price distance and book-level rank, respectively. At the individual-metric level, ReLOBGen achieves lower point estimates than LOBS5 on 18 of 21 metrics for GOOG, with statistically significant improvements on 16 of those 18. On INTC, it achieves lower point estimates on 20 of 21 metrics, with statistically significant improvements on 18 of those 20. Statistical significance is assessed using non-overlapping 99% bootstrap confidence intervals. Detailed L_1 and Wasserstein results are shown in Figures 5 and 6, respectively, with metric-group averages for Wasserstein distances reported in Table 12, all in Appendix C.3.

Table 4 shows that ReLOBGen achieves lower average conditional and market-impact discrepancies than LOBS5 on both stocks, reducing conditional averages (0.41 to 0.32 on GOOG; 0.20 to 0.10 on INTC) and market-impact averages (10.6 to 5.2 on GOOG; 5.6 to 4.5 on INTC). Figure 3 (a) further illustrates the difference in lagged price-response curves: LOBS5 tends to produce largely flat responses, whereas ReLOBGen better captures those observed in the real data. Regarding error accumulation, Figure 3 (b) shows lower errors for ReLOBGen on the two illustrated metrics, suggesting potential for more stable generation over longer rollout horizons. ReLOBGen also better preserves the marginal event-type distribution, reducing the total variation distance from 9.6 to 6.2 percentage points on GOOG, as detailed in Table 14 in Appendix C.4. Additional LOB-Bench evaluation results, including conditional Wasserstein results (Table 13), the full market-impact response curves (Figure 7), and error accumulation for all metrics (Figures 8 and 9), are provided in Appendix C.3.

5.4 Ablation studies

Effect of learned reference selection. Although uniform selection generates replayable messages, it does not necessarily improve rollout realism. As shown in Table 3, it only slightly reduces the overall L_1 distance relative to LOBS5 (ref.-last) on GOOG and even increases it on INTC, from 0.22 to 0.30. Thus, which eligible order is selected matters for rollout realism.

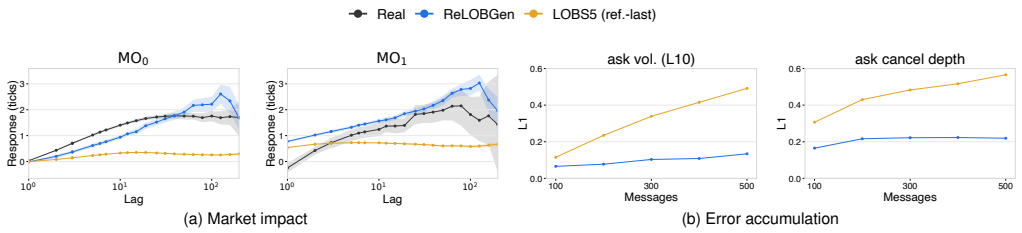


Figure 3: **LOB-Bench evaluation: market-impact response curves and error accumulation on GOOG.** (a) Responses after market orders without (MO_0) and with (MO_1) an immediate mid-price change; shading shows 99% bootstrap confidence intervals. (b) L_1 distance over successive 100-message intervals.

Table 5: **Teacher-forced NLLs under reference-last and reference-first token orders.** For fields represented by multiple tokens, NLLs are summed across tokens. Shaded columns highlight the price and size fields of the reference and event orders. Lower is better.

			E_t		R_t				X_t			
Ticker	Token order	Overall	Type	Side	Price	Size	Time (s)	Time (ns)	Price	Size	Δt (s)	Δt (ns)
GOOG	Ref.-last	19.01	0.76	0.51	0.00	0.02	0.29	1.33	1.83	1.29	0.00	12.96
	Ref.-first	18.94	0.75	0.51	0.89	0.55	0.30	1.29	0.96	0.74	0.00	12.97
INTC	Ref.-last	18.50	0.71	0.51	0.00	0.06	0.38	1.65	0.97	1.95	0.01	12.27
	Ref.-first	18.51	0.72	0.50	0.46	0.82	0.37	1.66	0.51	1.20	0.01	12.28

Tables 3 and 4 show that learned selection improves overall rollout realism over uniform selection on both stocks. Learned selection improves reference-order choices for cancel and delete events, with lower L_1 distances on all five related metrics for both stocks, as shown in Figure 5 in Appendix C.3: log time-to-cancel and the bid- and ask-side cancellation depth and level distributions. The gains are especially pronounced on INTC, where the top ten price levels of the LOB contain 2.7 times as many resting orders as in GOOG, consistent with the potential benefit of learned over uniform selection in larger eligible sets.

Effect of reference-first factorization. We evaluate field-wise teacher-forced NLLs under the two token orders on 10,000 windows from the test split. Table 5 shows similar overall NLLs and similar field-wise NLLs for fields other than price and size. The price and size NLLs suggest that reference-first ordering redistributes predictive uncertainty from the event-order fields to the reference-order fields that now precede them: reference-order NLLs increase, while the corresponding event-order NLLs decrease. This redistribution is consistent with the price and size dependencies in Table 1, which allow ground-truth reference-order fields to inform subsequent event-order prediction under teacher forcing. During inference, ReLOBGen selects an eligible resting order from a learned distribution and provides its current attributes before generating the event order, allowing subsequent generation to be conditioned and constrained by the selected reference. This use of the reference during generation may contribute to ReLOBGen's improved rollout realism.

6 Conclusion

We introduced ReLOBGen, a method for generating LOB messages that are replayable by construction against the current market state. We organized generation around two replayability conditions—reference eligibility and event-order compatibility—and satisfied them through a reference-first factorization that selects an eligible resting order before constraining event-order generation through inference-time masking. We further proposed a learned, efficiently computable distribution over eligible resting orders for realistic reference selection. Empirically, ReLOBGen achieved 100% replayability in 500-message rollouts on GOOG and INTC, improved most evaluated realism metrics over the LOBS5 baseline, and achieved a 2.7–3.6× speedup per replayed message. These results show that replayability by construction can be achieved alongside improved rollout realism and computational efficiency. Future work can build on ReLOBGen to train and evaluate trading strategies and analyze market impact in closed-loop market simulation.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology, assist with method implementation, and clean and reformat datasets. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, develop or write proofs, propose or refine hypotheses, assist with translation, perform qualitative or thematic data analysis, or interpret results. Additionally, we used generative AI tools for literature search, refinement of author-written text, and graph creation. We have reviewed all AI-assisted work. AI-assisted code passed unit tests, and AI-assisted writing was reviewed line by line. We also consulted original sources before citing them. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work is intended to support research on limit order book message generation and market simulation. The generated outputs should not be interpreted as financial advice. Any application to real-world trading or financial decision-making would require independent validation and appropriate risk assessment.

We used market data purchased from Databento[1].

REPRODUCIBILITY STATEMENT

To support reproducibility, we describe data preprocessing, model architectures, training configurations, and the rollout protocol in Appendix B.

REFERENCES

Alfred Backhouse, Kang Li, Jakob Foerster, Anisoara Calinescu, and Stefan Zohren. Painting the market: generative diffusion models for financial limit order book simulation and forecasting. *arXiv preprint arXiv:2509.05107*, 2025.

Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy. Hawkes processes in finance. *Market Microstructure and Liquidity*, 1(01):1550005, 2015.

David Byrd, Maria Hybinette, and Tucker Hybinette Balch. ABIDES: Towards high-fidelity multi-agent market simulation. In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2020.

Rama Cont, Sasha Stoikov, and Rishi Talreja. A stochastic model for order book dynamics. *Operations research*, 58(3):549–563, 2010.

Sascha Yves Frey, Kang Li, Peer Nagy, Silvia Sapora, Christopher Lu, Stefan Zohren, Jakob Foerster, and Anisoara Calinescu. JAX-LOB: A gpu-accelerated limit order book simulator to unlock large scale reinforcement learning for trading. In *ICAIF*, 2023.

Ruihong Huang and Tomas Polak. LOBSTER: Limit order book reconstruction system. *Available at SSRN 1977207*, 2011.

Weibing Huang, Charles-Albert Lehalle, and Mathieu Rosenbaum. Simulating and analyzing order book data: The queue-reactive model. *Journal of the American Statistical Association*, 110(509): 107–122, 2015.

Jiwon Jung and Kiseop Lee. Attention-based reading, highlighting, and forecasting of the limit order book. *Quantitative Finance*, 2025.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP*, 2020.

Maxime Kawawa-Beaudan, Srijan Sood, Kassiani Papasotiriou, Daniel Borrajo, and Manuela Veloso. TradeFM: A generative foundation model for trade-flow and market microstructure. *arXiv preprint arXiv:2602.23784*, 2026.

Junjie Li, Yang Liu, Weiqing Liu, Shikai Fang, Lewen Wang, Chang Xu, and Jiang Bian. MarS: a financial market simulation engine powered by generative foundation model. In *ICLR*, 2025.

Peer Nagy, Sascha Frey, Silvia Sapora, Kang Li, Anisoara Calinescu, Stefan Zohren, and Jakob Foerster. Generative ai for end-to-end limit order book modelling: A token-level autoregressive generative model of message flow using a deep state space network. In *ICAIF*, 2023.

Peer Nagy, Sascha Frey, Kang Li, Bidipta Sarkar, Svitlana Vyetrenko, Stefan Zohren, Ani Calinescu, and Jakob Foerster. LOB-Bench: Benchmarking generative ai for finance–an application to limit order book data. In *ICML*, 2025.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *EMNLP*, 2021.

Eric Smith, J Doyne Farmer, László Gillemot, and Supriya Krishnamurthy. Statistical theory of the continuous double auction. *Quantitative Finance*, 3(6):481–514, 2003.

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. In *ICLR*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Aaron Wheeler and Jeffrey D Varner. MarketGPT: Developing a pre-trained transformer (gpt) for modeling financial time series. *arXiv preprint arXiv:2411.16585*, 2024.

Zihao Zhang, Stefan Zohren, and Stephen Roberts. DeepLOB: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 2019.

A POST-PROCESSING IN PRIOR METHODS

Prior LOB message generators do not guarantee that their raw outputs can be replayed on the current LOB. Hence, their market-simulation pipelines rely on model-specific, often heuristic, post-processing to convert raw outputs into simulator actions.

The design of this post-processing depends on whether a generated message is treated as an exchange-side record of an event that has already occurred or as a trader-side instruction submitted to the matching engine. LOBS5 corrects generated messages into exchange-side records, whereas MarS generates trader-side instructions. MarketGPT incorporates aspects of both designs.

A.1 LOBS5 (NAGY ET AL., 2023)

LOBS5 uses the generated-message field layout $\widehat{M}_t = (\widehat{E}_t, \widehat{X}_t, \widehat{R}_t)$, with the reference fields appearing last. Here, $\widehat{E}_t$ denotes event type and side, $\widehat{X}_t$ denotes the event order, and $\widehat{R}_t$ denotes the referenced order.

$\widehat{E}_t[\text{type}] = \text{Add}$.

- **Rejection:** LOBS5 rejects the message if $\widehat{X}_t[\text{price}]$ is marketable against the opposite best price.
- **Correction:** Otherwise, LOBS5 replaces any generated non-N/A reference field in $\widehat{R}_t$ with N/A. No other correction is applied.

$\widehat{E}_t[\text{type}] \in \{\text{Cancel}, \text{Delete}\}$.

- **Rejection:** LOBS5 matches a resting order $o \in \mathcal{O}_{t-1}$ to $\widehat{R}_t$ under $(\text{price}, \text{size}, \text{time}) \to (\text{price}, \text{size}) \to (\text{price})$. The final price-only fallback applies only to INIT liquidity, which represents the initial book state at each side–price level as a single order. The message is rejected if no such o is found.
- **Correction:** If such an order o is found, LOBS5 sets

$$R_t = o, \quad X_t[\text{price}] = R_t[\text{price}], \quad X_t[\text{size}] = \min\{\widehat{X}_t[\text{size}], R_t[\text{size}]\}.$$

  It sets $E_t[\text{type}]$ to Cancel if $X_t[\text{size}] < R_t[\text{size}]$, and to Delete otherwise.
- **Our reproduction:** We first attempt an exact match against the reconstructed resting-order set, then fall back to full and relaxed matching within the message history, following the original code. We use exact resting-order initialization and compare it with INIT-order initialization in Appendix C.2.

$\widehat{E}_t[\text{type}] = \text{Execute}$.

- **Rejection:** The message is rejected if no executable order is found.
- **Correction:** Otherwise, LOBS5 ignores $\widehat{R}_t$ and sets $R_t = \text{FrontOfQueue}(\mathcal{O}_{t-1}^{(\widehat{E}_t[\text{side}])})$. It then sets $X_t[\text{price}] = R_t[\text{price}]$ and $X_t[\text{size}] = \min\{\widehat{X}_t[\text{size}], R_t[\text{size}]\}$.

A.2 MARKETGPT (WHEELER & VARNER, 2024)

MarketGPT uses the generated-message field layout $\widehat{M}_t = (\widehat{E}_t, \widehat{X}_t, \widehat{R}_t)$, with an additional $\widehat{X}_t[\text{remaining size}]$ field and five event types: Add, Execute, Execute-at-different-price, Cancel/Delete, and Replace. Execute-at-different-price is unsupported in the released generated rollouts, so we omit it below. For Cancel/Delete, $\widehat{X}_t[\text{size}]$ and $\widehat{X}_t[\text{remaining size}]$ denote the canceled and remaining quantities, respectively.

$\widehat{E}_t[\text{type}] = \text{Add}$**.**

- **Simulator handling:** The generated message is submitted directly as a limit order without correction or rejection. If it is marketable, the matching engine executes it against the opposite book in price–time priority, and any unfilled quantity rests at the generated limit price.

$\widehat{E}_t[\text{type}] = \text{Execute}$**.**

- **Simulator handling:** MarketGPT ignores $\widehat{R}_t$ and $\widehat{X}_t[\text{price}]$ and treats the generated side and size as a market order. It repeatedly matches opposite-side resting orders in price–time priority until the generated size is filled or the opposite book is exhausted, discarding any unfilled quantity.

$\widehat{E}_t[\text{type}] \in \{\text{Cancel}, \text{Delete}\}$**.**

- **Rejection:** MarketGPT matches a resting order $o \in \mathcal{O}_{t-1}$ to $(\widehat{X}_t[\text{price}], \widehat{X}_t[\text{size}] + \widehat{X}_t[\text{remaining size}], \widehat{R}_t[\text{time}])$ under $(\text{price}, \text{size}, \text{time}) \to (\text{price}, \text{size}) \to (\text{price}, \text{time}) \to (\text{price})$. The final price-only fallback selects the front-of-queue resting order at $\widehat{X}_t[\text{price}]$. The message is rejected if no such o is found.
- **Correction:** If such an order o is found, let $\widehat{\rho}_t = \widehat{X}_t[\text{size}]/(\widehat{X}_t[\text{size}] + \widehat{X}_t[\text{remaining size}])$ denote the generated cancellation ratio. MarketGPT sets
$$R_t = o, \quad X_t[\text{size}] = \lfloor R_t[\text{size}]\widehat{\rho}_t \rfloor, \quad X_t[\text{remaining size}] = R_t[\text{size}] - X_t[\text{size}],$$
preserving the generated cancellation ratio for the matched resting order up to integer rounding.

$\widehat{E}_t[\text{type}] = \text{Replace}$**.**

- **Rejection:** MarketGPT matches a resting order $o \in \mathcal{O}_{t-1}$ to $\widehat{R}_t$ under $(\text{price}, \text{size}, \text{time}) \to (\text{price}, \text{size}) \to (\text{price}, \text{time}) \to (\text{price})$. The final price-only fallback selects the front-of-queue resting order at $\widehat{R}_t[\text{price}]$. The message is rejected if no such o is found.
- **Correction:** If such an order o is found, MarketGPT sets $R_t = o$ and retains $X_t = \widehat{X}_t$.

A.3 MarS (Li et al., 2025)

In our notation, MarS represents each generated output as $\widehat{M}_t = (\widehat{E}_t, \widehat{X}_t)$ without reference-order fields, where $\widehat{E}_t \in \{\text{Bid}, \text{Ask}, \text{Cancel}\}$. It interprets each output as a trader-side instruction to the simulated clearing house rather than as a complete exchange message.

$\widehat{E}_t \in \{\text{Bid}, \text{Ask}\}$**.**

- **Simulator handling:** MarS submits the generated Bid or Ask directly as a limit order without correction or rejection. If it is marketable, the simulated clearing house executes it against opposite-side resting orders, and any unfilled quantity rests at the generated price.

$\widehat{E}_t = \text{Cancel}$**.**

- **Rejection:** If no resting order is available for cancellation, MarS discards the generated cancellation by returning an empty simulator action. The discarded output is not added to the model's order history, and the agent immediately samples again without rolling back the elapsed simulation time.
- **Correction:** MarS projects the generated cancellation onto the current resting-order set $\mathcal{O}_{t-1}$ by selecting an available resting order $o \in \mathcal{O}_{t-1}$ whose price $o[\text{price}]$ is closest to $\widehat{X}_t[\text{price}]$. It assigns actual order IDs and sides and distributes $\widehat{X}_t[\text{size}]$ across the resting orders at $o[\text{price}]$, capped by their available size.

B EXPERIMENTAL DETAILS

B.1 PREPROCESSING

We convert XNAS.ITCH MBO data for GOOG and INTC from Databento[2] into LOBSTER-compatible messages (Huang & Polak, 2011) and follow the preprocessing pipeline of LOBS5 (Nagy et al., 2023), except for the reference-order representation. See below for details.

Databento MBO to LOBSTER-compatible message. Within each stream identified by `(publisher_id, instrument_id, channel_id)`, consecutive records in feed order with the same `sequence` value form an event group, which is converted according to Table 6. Under these conversion rules, individual fields are mapped as shown in Table 7, with format conversions applied as needed.

Table 6: **Conversion of Databento MBO action patterns to LOBSTER-compatible messages.** Reset clears the order state once at session start without emitting a message. Sole trades (single-`T` groups with `side=N`) may correspond to Type 5 but are omitted because their side is unknown.

MBO action pattern	LOBSTER output	Side source
Add (`A`)	Type 1: Add	side of `A`
Cancel (`C`)	Type 2/3: Cancel/Delete	side of `C`
Replace (`C` → `A`)	Type 2/3 → Type 1	side of each `C` / `A`
Visible execution (`T` → `F` → `C`)	Type 4: Execute	side of `F`
Cross trade (`F` → `C`)	Type 6: Cross trade	side of `C`
Reset (`R`)	No LOBSTER message	—
Sole trade (`T` with `side=N`)	No LOBSTER message	—

Table 7: **Field correspondence between Databento MBO and LOBSTER-compatible messages.**

MBO field	LOBSTER-compatible field	Description
`ts_event`	`Time`	Exchange-recorded event timestamp
`action`	`Type`	Order-book event category
`order_id`	`Order ID`	Individual order identifier
`side`	`Direction`	Buy or sell side
`price`	`Price`	Order or execution price
`size`	`Size`	Order, cancellation, or execution quantity

Message representation. Table 8 summarizes normalization and encoding in ReLOBGen. All fields share a single vocabulary of 12,011 tokens, including three special tokens for masking, hiding, and not-applicable values. The event order's absolute timestamp is determined by adding its inter-arrival time to the previous event's timestamp, rather than generated by the model.

LOBS5 (Nagy et al., 2023) copies the encoded field values from the referenced order's original Add message into the reference-order fields, retaining the price encoding relative to the mid-price at submission rather than the current mid-price.

[2] https://databento.com/portal/catalog/us-equities#XNAS.ITCH

Table 8: **Normalization and encoding in ReLOBGen.** Prices are encoded as tick offsets from the tick-aligned mid-price, clipped to $[-999, 999]$: $\Delta p = \mathrm{clip}((p_{\mathrm{order}} - p_{\mathrm{mid}})/\tau, -999, 999)$. Here, $p_{\mathrm{mid}} = \tau\lfloor(p_{\mathrm{ask},1} + p_{\mathrm{bid},1})/(2\tau)\rfloor$, $p_{\mathrm{ask},1}$ and $p_{\mathrm{bid},1}$ denote the best ask and bid prices, respectively, and $\tau = 100$. Each time component is encoded in base 1000, with one token per three-digit group.

Field	# Tokens	Normalization and encoding
Event type	1	No normalization
Event side	1	Ask: 0 / Bid: 1
Reference-order price	2	$(\mathrm{sign}(\Delta p), \lvert\Delta p\rvert)$
Reference-order size	1	$\min(\text{remaining size}, 9999)$
Reference-order time (s)	2	Seconds component
Reference-order time (ns)	3	Nanoseconds component
Event-order price	2	$(\mathrm{sign}(\Delta p), \lvert\Delta p\rvert)$
Event-order size	1	$\min(\text{size}, 9999)$
Event-order Δt (s)	1	Seconds component
Event-order Δt (ns)	3	Nanoseconds component
Event-order time (s)	2	Seconds component
Event-order time (ns)	3	Nanoseconds component

Book representation. We reconstruct book states by replaying the message sequence. Then, following LOBS5 (Nagy et al., 2023), we represent each book state as a 501-dimensional continuous vector comprising the mid-price change in ticks and signed volumes on a 500-slot price grid centered on the tick-aligned mid-price:

$$\mathbf{b}_t = (\Delta p_{\mathrm{mid},t}/\tau, x_{t,0}, \ldots, x_{t,499}) \in \mathbb{R}^{501}, \tag{6}$$

where $\Delta p_{\mathrm{mid},t} = p_{\mathrm{mid},t} - p_{\mathrm{mid},t-1}$ and $\tau = 100$. We map the top 10 ask and bid levels onto this grid by their tick offsets from the mid-price. Each slot stores the total quantity divided by 1000, negative for asks and positive for bids. Levels outside the grid are omitted, and empty slots are zero.

Filtering. For training, we retain only Type 1 (Add), Type 2 (Cancel), Type 3 (Delete), and Type 4 (Execute) messages whose prices lie between the 10th-best bid and 10th-best ask prices at the time of the event. We use only 500-message sequences fully contained within regular trading hours (09:30–16:00 ET).

B.2 Training configuration

We follow the scaled-up LOBS5 architecture used in LOB-Bench (Nagy et al., 2023; 2025) and likewise use one year of training data for both stocks, covering Jan. 1–Dec. 31, 2025, with Jan. 1–15, 2026 for validation. We train the message generation model π_θ first, followed by the query and order projection heads f_{query} and f_{order} on the same data periods. Table 9 summarizes the training settings, and Table 10 details the architecture of each component. Training budgets count message windows for π_θ and window–mid-price pairs for the projection heads, with 256 mid-prices per window.

Table 9: **Training configuration.** In both training stages, learning rates use linear warmup over the first 10% of training steps, followed by cosine decay to 10% of their peak values.

Training stage	Dataset	Budget	Batch size	Optimizer	Peak LR	GPU hours
Base model	GOOG	100M	512	Adam	5×10^{-4}	8 L40S × 54 h
	INTC	50M	512		5×10^{-4}	8 L40S × 27 h
Reference selection	GOOG	200M	256	AdamW	1×10^{-4}	1 A6000 × 33 h
	INTC	100M	256		1×10^{-4}	1 A6000 × 25 h

Table 10: **Model architecture.** π_θ is the message generation model; f_{query} and f_{order} are the query and order projection heads, respectively.

Model	# Params	Details
π_θ	35.78M	• S5 backbone: $d_{\text{model}} = 512$; SSM state size 512. • 2 S5 message-encoder layers; 1 pre- and 1 post-book-encoder layer. • 12 S5 layers for message–book fusion.
f_{query}	0.13M	• Layer-normalized 512-d state concatenated with a GELU-projected mid-price displacement ($1 \to 512$). • $1024 \to 128$ projection followed by layer normalization.
f_{order}	2.16M	• Eight 512-d token embeddings flattened into a $4096 \to 512 \to 128$ MLP, with GELU after the first layer and output layer normalization.

B.3 Rollout protocol

For each stock, we use 100 rollout windows from each of 10 test days, January 16–30, 2026. Each rollout starts from the exact resting-order set within the top 20 price levels on each side, providing a buffer of deeper resting orders that may enter the top 10 price levels during the rollout. Consistent with training, model LOB-state inputs and eligible reference sets are restricted to the current top 10 price levels on each side. We do not apply temperature scaling or top-k/top-p filtering during either resting-order selection or token sampling. To avoid spending excessive computation on stalled rollouts, we restart a rollout with a different random seed after 100 consecutive failed attempts.

C Additional experimental results

C.1 Event-wise replayability

Table 11 reports post-processing and replayability violation rates by event type on GOOG and INTC. Both LOBS5 baselines rarely require post-processing for add events, but frequently reject cancel and delete events and correct all execute events. Across non-add event types, reference violations are more frequent than event-order violations on both stocks. ReLOBGen records no violations and requires no post-processing for any event type.

Table 11: **Event-wise post-processing rates and replayability violations.** Positive rates below 0.1% are reported as <0.1%. Lower is better, and bold indicates the lowest value.

				Post-processing		Replayability violations	
Ticker	Event	Method	Attempts	Correction	Rejection	Reference	Event-order
GOOG	Add	LOBS5 (ref.-last)	255,720	**0.0%**	0.2%	**0.0%**	0.2%
		LOBS5 (ref.-first)	259,199	**0.0%**	<0.1%	**0.0%**	<0.1%
		ReLOBGen	243,723	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	Cancel	LOBS5 (ref.-last)	4,692	9.9%	67.6%	76.1%	5.1%
		LOBS5 (ref.-first)	2,781	9.4%	67.7%	76.7%	1.8%
		ReLOBGen	1,058	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	Delete	LOBS5 (ref.-last)	353,723	8.9%	47.8%	56.7%	<0.1%
		LOBS5 (ref.-first)	379,291	8.0%	50.1%	58.1%	<0.1%
		ReLOBGen	209,395	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	Execute	LOBS5 (ref.-last)	58,814	100.0%	**0.0%**	100.0%	3.1%
		LOBS5 (ref.-first)	50,781	100.0%	**0.0%**	100.0%	0.9%
		ReLOBGen	45,824	**0.0%**	**0.0%**	**0.0%**	**0.0%**
INTC	Add	LOBS5 (ref.-last)	264,173	<0.1%	0.7%	<0.1%	0.7%
		LOBS5 (ref.-first)	269,847	**0.0%**	0.2%	**0.0%**	0.2%
		ReLOBGen	253,908	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	Cancel	LOBS5 (ref.-last)	1,494	10.8%	66.9%	73.8%	13.9%
		LOBS5 (ref.-first)	1,377	19.0%	50.5%	65.4%	13.8%
		ReLOBGen	1,068	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	Delete	LOBS5 (ref.-last)	311,405	12.1%	30.3%	42.4%	0.1%
		LOBS5 (ref.-first)	291,193	18.0%	25.6%	43.6%	<0.1%
		ReLOBGen	223,952	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	Execute	LOBS5 (ref.-last)	19,952	100.0%	**0.0%**	100.0%	3.4%
		LOBS5 (ref.-first)	13,473	100.0%	**0.0%**	100.0%	2.3%
		ReLOBGen	21,072	**0.0%**	**0.0%**	**0.0%**	**0.0%**

C.2 Effect of resting-order initialization on replayability

We compare two ways of initializing the resting-order set at the start of a rollout. INIT-order initialization, as used in the original LOBS5 implementation, represents the total volume at each side–price level of the initial LOB state as a single INIT order, whereas exact resting-order initialization preserves the individual orders reconstructed from market-by-order data. INIT orders do not correspond to individual orders in the model's training messages and can only be matched to generated references through price-only matching during post-processing. Our evaluation uses exact resting-order initialization because the exact resting-order set is available from our data.

Figure 4 shows that LOBS5 (ref.-last) still requires correction and rejection under INIT-order initialization on GOOG. With INIT orders, rejection rates start lower but approach those under exact initialization by the end of the rollout. Together with the higher correction rates, this pattern suggests that the initial reduction in rejection reflects the looser matching available for INIT orders, rather than improved replayability of the raw generated messages.

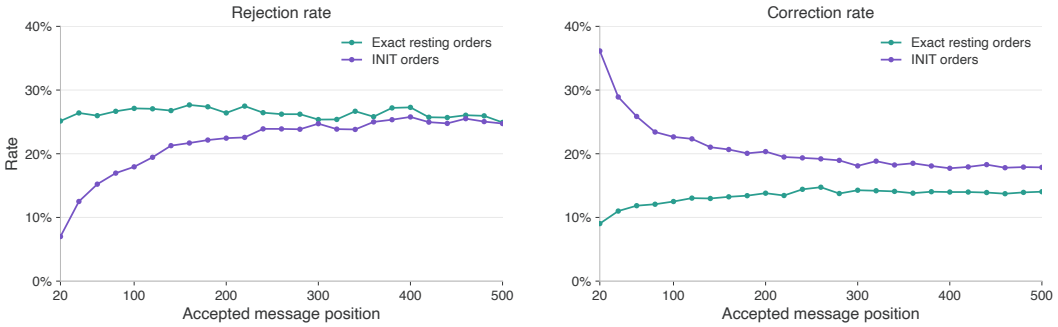


Figure 4: **Effect of resting-order initialization on replayability.** We report rejection and correction rates for LOBS5 (ref.-last) on GOOG under INIT-order and exact resting-order initialization. Rates are computed over generation attempts grouped into 20-message bins by accepted-message position.

C.3 ADDITIONAL LOB-BENCH RESULTS

Unconditional distributions. ReLOBGen achieves broadly lower unconditional L_1 and Wasserstein distances than LOBS5 across metric groups (Tables 3 and 12). At the individual-metric level, ReLOBGen achieves lower Wasserstein point estimates on 16 of 21 metrics for GOOG, with statistically significant improvements on 14 of those 16. On INTC, it achieves lower point estimates on 20 of 21 metrics, with statistically significant improvements on 18 of those 20, using the same CI non-overlap criterion as in Section 5.3. Figures 5 and 6 show the individual metric distances and their 99% bootstrap confidence intervals for L_1 and Wasserstein distances, respectively.

Table 12: **LOB-Bench evaluation: metric-group summary of unconditional Wasserstein distances.** Overall averages all metrics, while each metric-group column averages the metrics in its group. Lower is better, and bold indicates the lowest value.

			Metric group					
Ticker	Method	Overall	State	Times	Volumes	Depths	Levels	Trades
GOOG	LOBS5 (ref.-last)	0.43	0.59	0.31	0.34	0.69	0.37	0.35
	Ref.-first + uniform	0.33	0.39	0.33	0.23	0.28	0.44	0.32
	ReLOBGen	**0.22**	**0.33**	**0.20**	**0.19**	**0.19**	**0.14**	**0.28**
INTC	LOBS5 (ref.-last)	0.30	0.34	**0.26**	0.14	0.40	0.34	0.29
	Ref.-first + uniform	0.46	0.23	0.64	0.28	0.51	0.67	0.42
	ReLOBGen	**0.15**	**0.09**	0.37	**0.11**	**0.11**	**0.10**	**0.18**

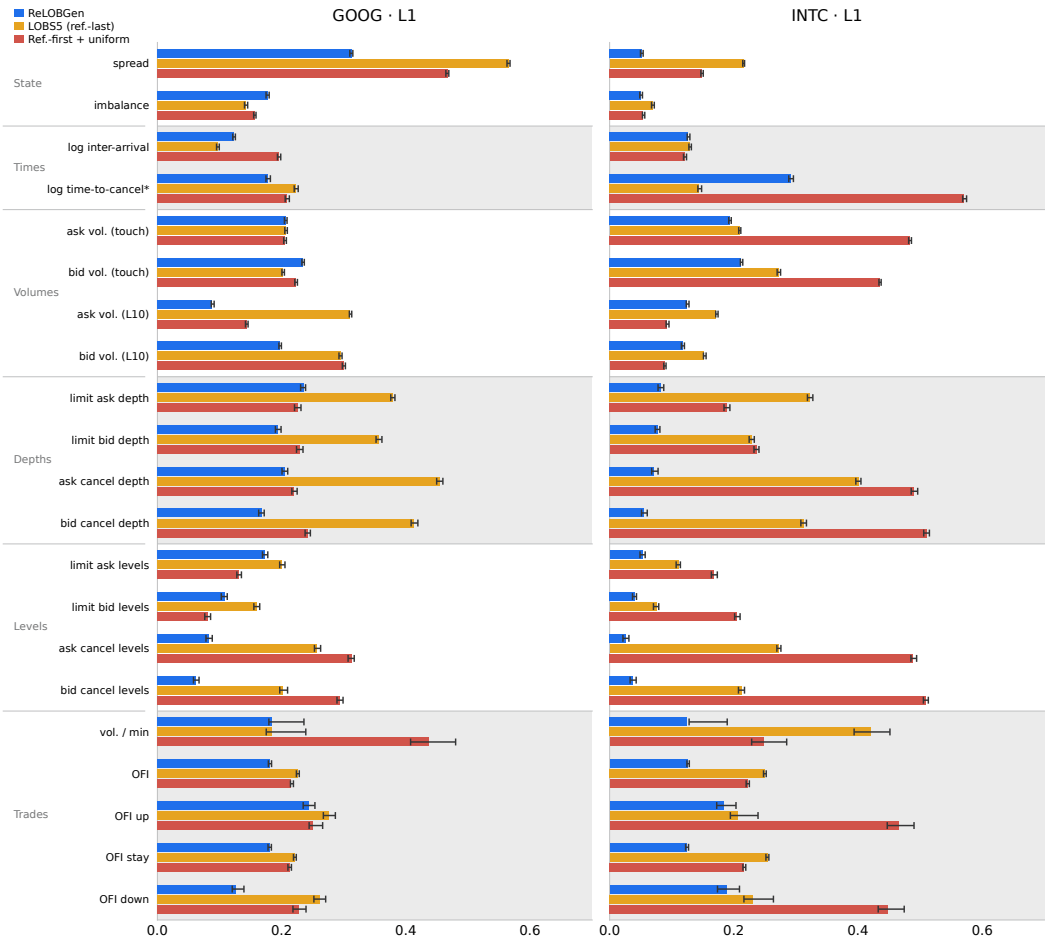


Figure 5: **LOB-Bench evaluation: unconditional L_1 distances.** Log time-to-cancel* measures time-to-cancel for all orders canceled during the rollout using exact resting-order information, including cancellations of orders present in the initial LOB that were excluded from the original LOB-Bench implementation. Error bars indicate 99% percentile bootstrap confidence intervals. Lower is better.

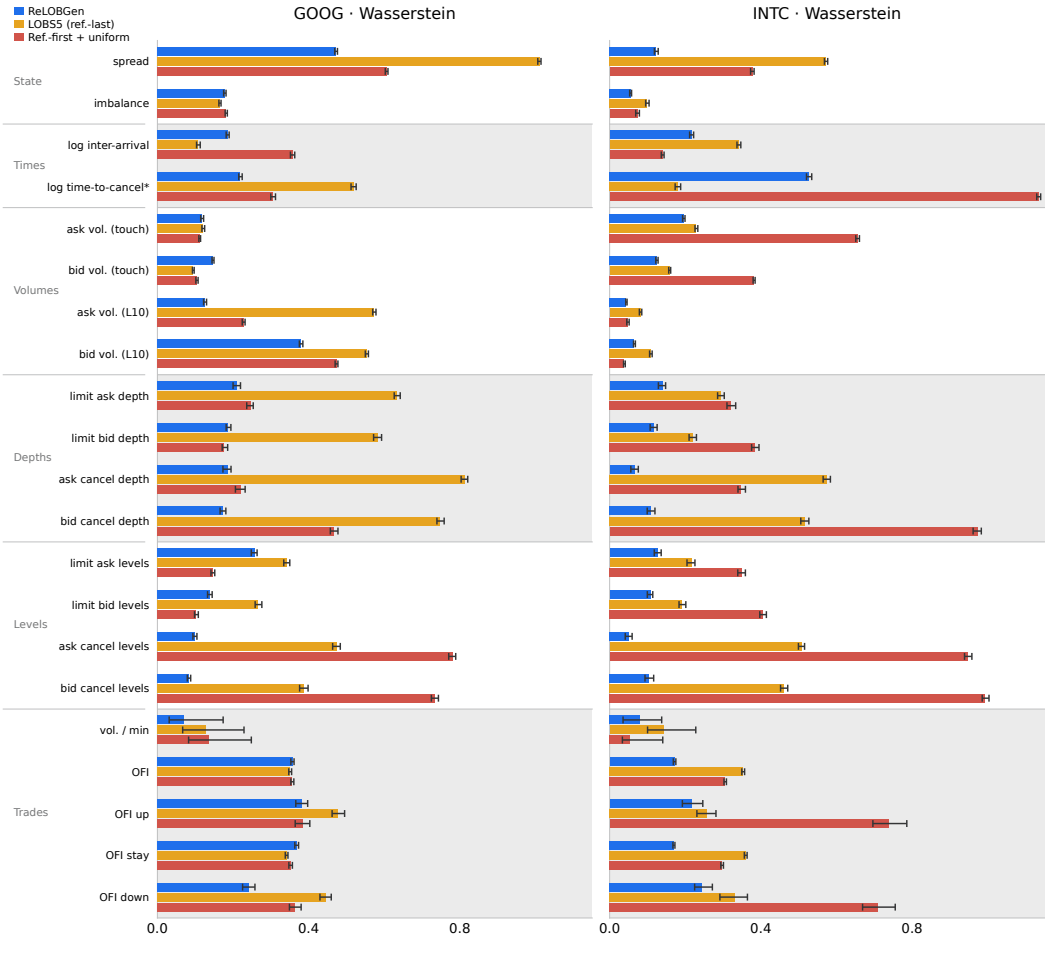


Figure 6: **LOB-Bench evaluation: unconditional Wasserstein distances.** Log time-to-cancel* measures time-to-cancel for all orders canceled during the rollout using exact resting-order information, including cancellations of orders present in the initial LOB that were excluded from the original LOB-Bench implementation. Error bars indicate 99% percentile bootstrap confidence intervals. Lower is better.

Conditional distributions. ReLOBGen achieves lower conditional Wasserstein point estimates than LOBS5 on 5 of 6 metrics across the two stocks (Table 13).

Table 13: **LOB-Bench evaluation: conditional Wasserstein distances.** Conditional 99% percentile bootstrap confidence intervals have half-widths below 0.012. Lower is better, and bold indicates the lowest value.

		Conditional (W_1)		
Ticker	Method	ask vol. | spread	spread | time	spread | volatility
GOOG	LOBS5 (ref.-last)	**0.12**	1.17	0.91
	Ref.-first + uniform	**0.12**	0.78	1.03
	ReLOBGen	0.17	**0.60**	**0.83**
INTC	LOBS5 (ref.-last)	0.24	0.57	0.50
	Ref.-first + uniform	0.67	0.36	0.23
	ReLOBGen	**0.21**	**0.12**	**0.17**

Market-impact responses. Figure 7 shows price-response curves following market orders, limit orders, and cancellations on GOOG and INTC. LOBS5 produces largely flat response curves across all six event types on GOOG and across the three event types without an immediate mid-price change on INTC. ReLOBGen more closely matches the real responses, with lower market-impact discrepancies than LOBS5 on 9 of 12 metrics across the two stocks (Table 4).

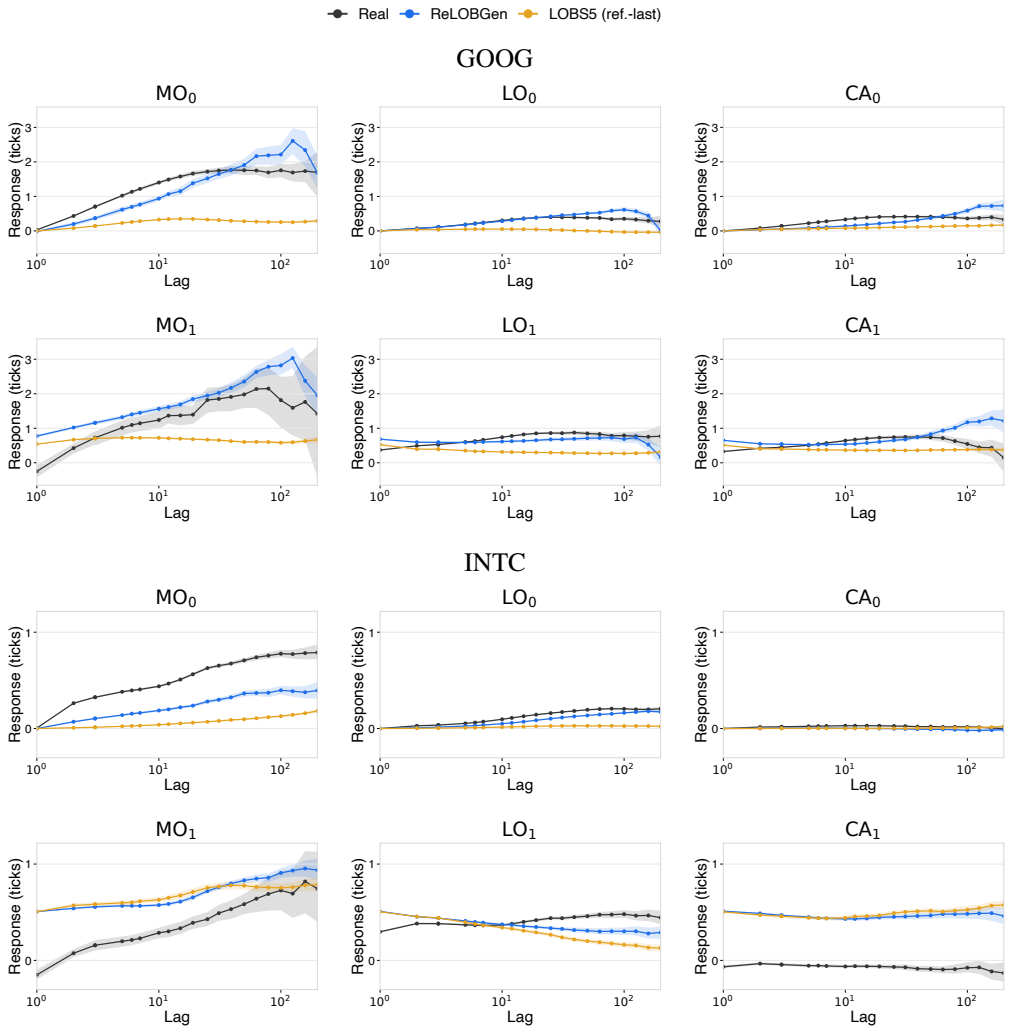


Figure 7: **LOB-Bench evaluation: market-impact response curves.** Responses are plotted against event lag for market orders (MO), limit orders (LO), and cancellations (CA), with subscripts 0 and 1 for events without and with an immediate mid-price change. Shaded regions indicate 99% bootstrap confidence intervals.

Error accumulation. Figures 8 and 9 show L_1 and Wasserstein divergences between generated and real distributions on GOOG and INTC at 100-message intervals. ReLOBGen generally exhibits lower divergence values and flatter slopes than LOBS5 over the rollout horizon.

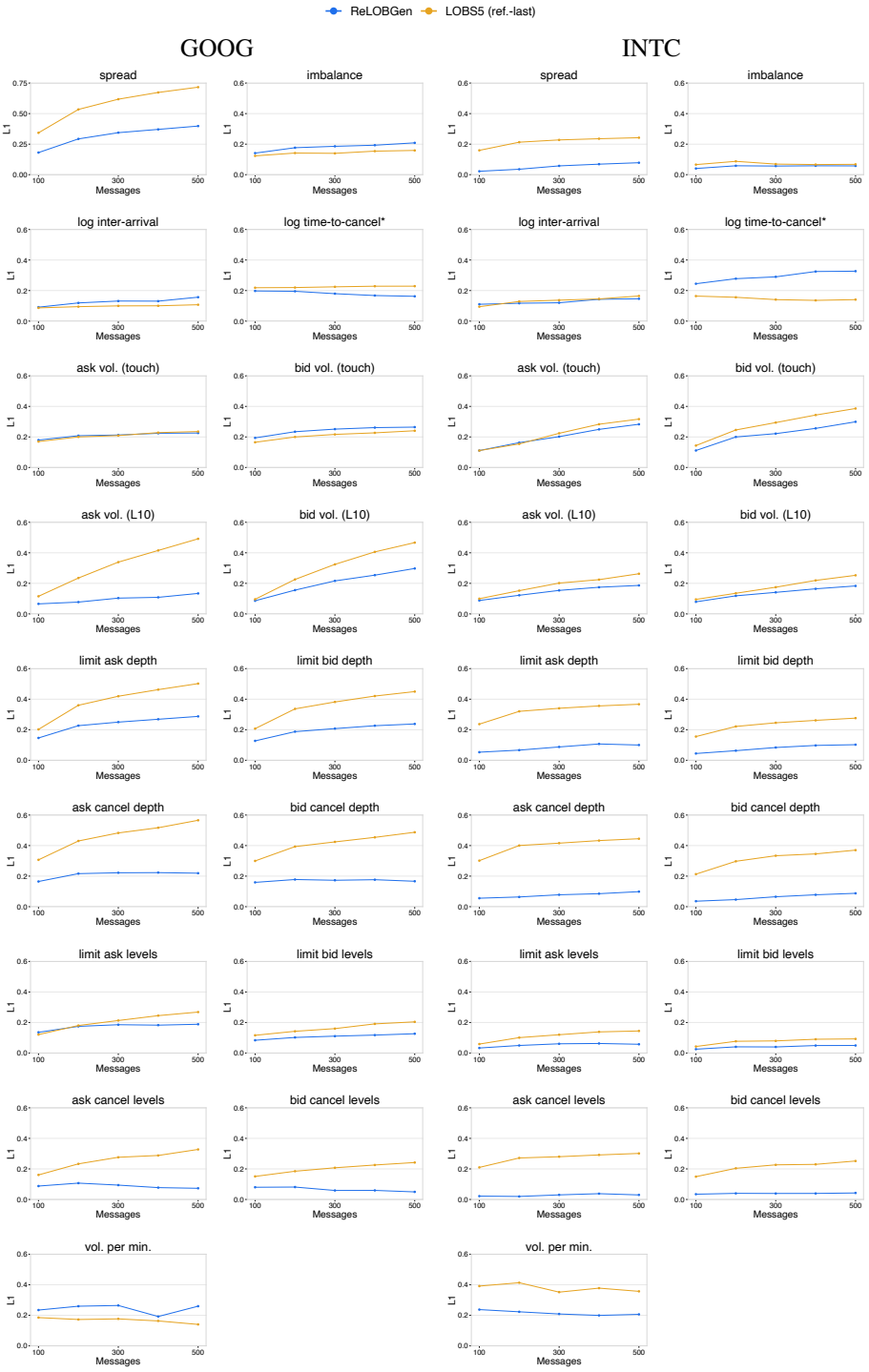


Figure 8: **LOB-Bench evaluation: error accumulation in L_1 distances.** Log time-to-cancel* measures time-to-cancel for all orders canceled during the rollout using exact resting-order information, including cancellations of orders present in the initial LOB that were excluded from the original LOB-Bench implementation. Lower is better.

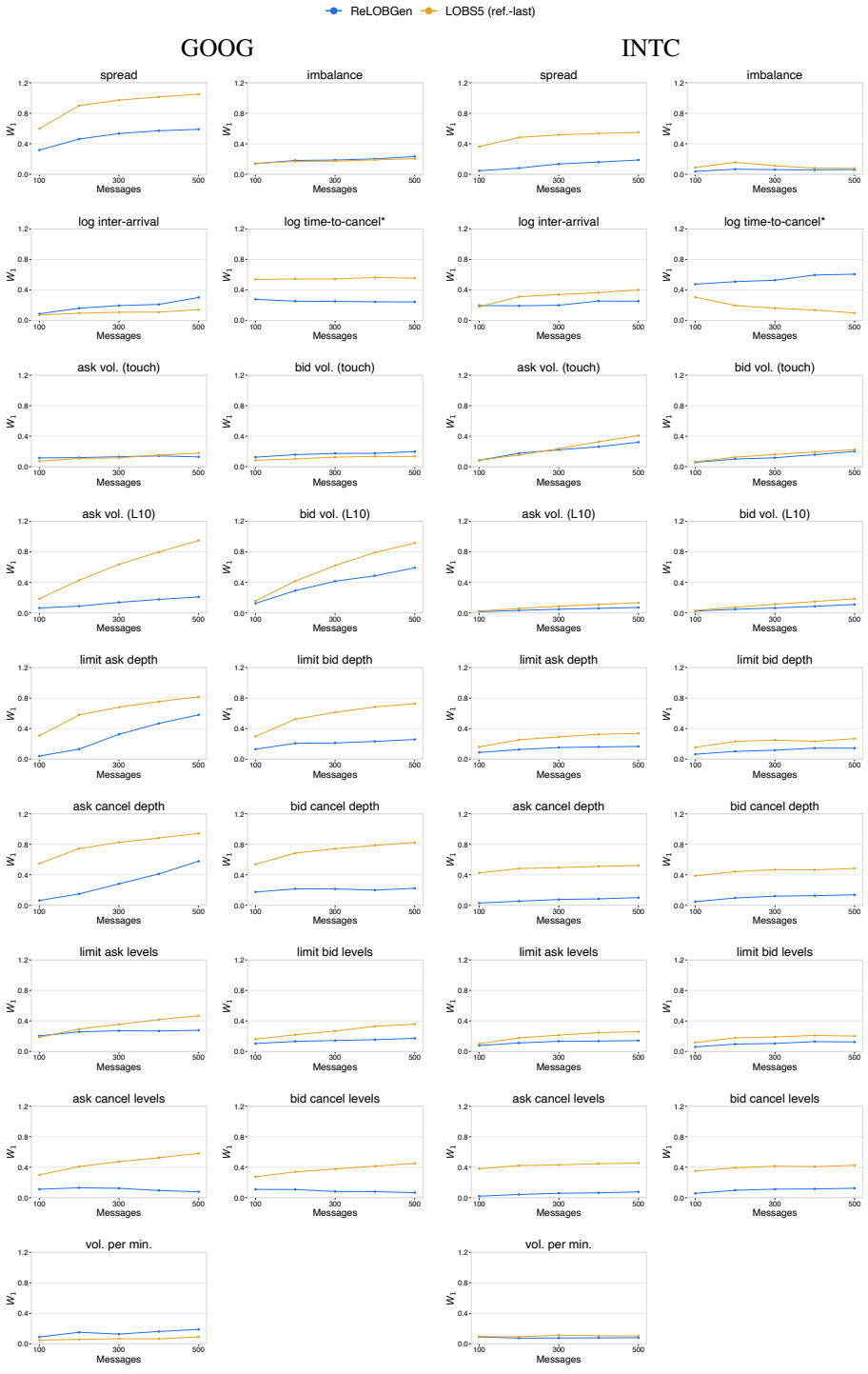


Figure 9: **LOB-Bench evaluation: error accumulation in Wasserstein distances.** Log time-to-cancel* measures time-to-cancel for all orders canceled during the rollout using exact resting-order information, including cancellations of orders present in the initial LOB that were excluded from the original LOB-Bench implementation. Lower is better.

C.4 MARGINAL EVENT-TYPE DISTRIBUTIONS

Table 14 compares marginal event-type distributions in actual and generated messages on GOOG and INTC. ReLOBGen achieves the lowest total variation distance among the evaluated methods on both stocks, indicating a closer match to the actual event-type distribution.

Table 14: **Marginal event-type frequencies in actual and generated messages.** TV denotes the total variation distance from the actual distribution, reported in percentage points. Lower TV is better, and bold indicates the lowest value.

Ticker	Method	Add	Cancel	Delete	Execute	TV
GOOG	Actual	50.2%	0.2%	46.6%	3.0%	–
	LOBS5 (ref.-last)	51.0%	0.3%	36.9%	11.8%	9.6 pp
	Ref.-first + uniform	45.2%	0.2%	38.4%	16.2%	13.2 pp
	ReLOBGen	48.7%	0.2%	41.9%	9.2%	**6.2 pp**
INTC	Actual	49.7%	0.2%	43.7%	6.4%	–
	LOBS5 (ref.-last)	52.5%	0.1%	43.4%	4.0%	2.8 pp
	Ref.-first + uniform	47.4%	0.2%	44.1%	8.3%	2.3 pp
	ReLOBGen	50.8%	0.2%	44.8%	4.2%	**2.2 pp**

D RESTING-ORDER COVERAGE BY HISTORY LENGTH

Figure 10 shows resting-order coverage as a function of model-history length for GOOG and INTC. Let $\mathcal{O}_{t-1}^{(10)}$ denote the subset of $\mathcal{O}_{t-1}$ within the current top 10 price levels on each side. We define coverage as

$$C_t(L) = \frac{\left|\left\{o \in \mathcal{O}_{t-1}^{(10)} : o \text{ appears in } M_{t-L:t-1}\right\}\right|}{|\mathcal{O}_{t-1}^{(10)}|}, \tag{7}$$

and report the mean of $C_t(L)$ across windows. An order appears in the history if its raw order ID occurs in the preceding L filtered messages.

At the 500-message context length used by our models, coverage is only 60.4% for GOOG and 28.6% for INTC. Coverage increases to 86.2% and 65.6%, respectively, with 5,000 preceding messages, with diminishing gains as history length grows. Even at this length, a substantial fraction of the current resting orders remains absent from the message history.

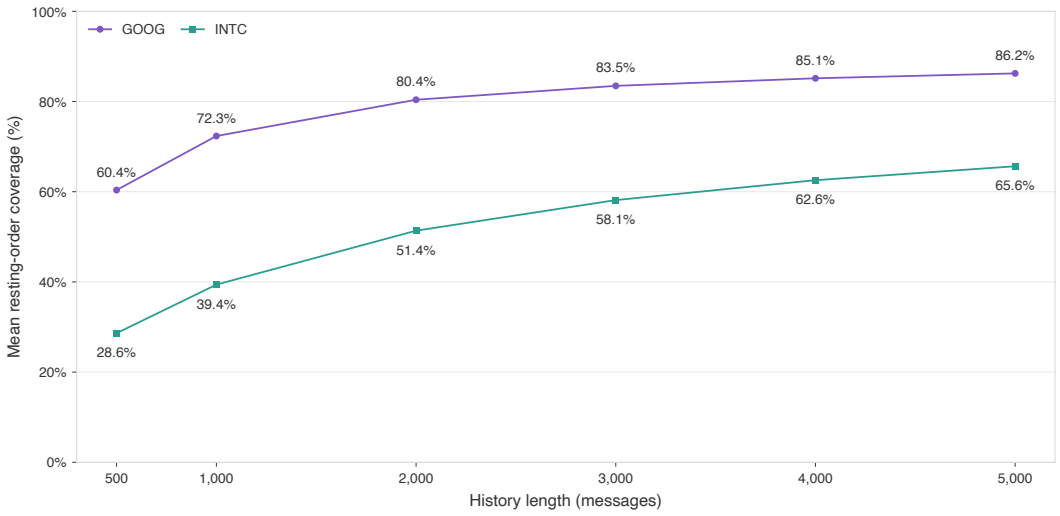


Figure 10: **Mean resting-order coverage by history length on GOOG and INTC.** Coverage is evaluated during regular trading hours on the test split.