Solving Optimal Execution Problems via In-Context Operator Networks

Tingwei Meng* Moritz Voss*† Nils Detering‡
Giulio Farolfi* Stanley Osher* Georg Menz*

February 10, 2026

Abstract

We propose a novel transformer-based neural network architecture (ICON-OCnet) for solving optimal order execution problems in the presence of unknown price impact. Our architecture facilitates data-driven in-context operator learning for the incurred price impact by merging offline pre-training with online few-shot prompting inference. First, the operator learning component (ICON) learns the prevailing price impact environment from only a few executed trade and price impact trajectories (time series data) provided as context. Second, we employ ICON as a surrogate operator to train a neural network policy (OCnet) for the optimal order execution strategy for the price impact regime inferred from the in-context examples. We study the efficiency of our approach for linear propagator models with path-dependent transient price impact and explicitly known optimal execution strategies. In this model class, price impact persists and decays over time according to some propagator kernel. We illustrate that ICON is capable of accurately inferring the underlying price impact model from the data prompts, even for propagator kernels not seen in the training data. Moreover, we demonstrate that ICON-OCnet correctly retrieves the exact optimal order execution strategy for the model generating the in-context examples. Our introduced methodology is very general, offering a new approach to solving path-dependent optimal stochastic control problems sample-based with unknown state dynamics.

1 Introduction

Devising optimal order execution strategies for buying or selling large volumes of shares of a stock on a centralized exchange is a major concern for large institutional investors and hence became a well-studied problem in quantitative finance in the past two decades. The aim is to split up a large meta order into smaller child orders which are executed over some time horizon to mitigate adverse price impact incurred by large trades. We refer to the monographs [CJP15, Gué16, BBDG18, Web23] for a comprehensive overview of this topic. Typically, these problems are addressed as optimal stochastic control problems by first formulating a price impact model that describes how trades affect the execution price, and then solving for an optimal execution strategy tailored to this model.

In contrast, our approach is concerned with the limitations of postulating a specific parametric model in real-world trading applications, where market conditions are non-stationary

*University of California Los Angeles, Department of Mathematics, Los Angeles, CA 90095, USA.
†Corresponding author.
‡Heinrich Heine University Düsseldorf, Mathematisches Institut, Universitätsstraße 1, 40225 Düsseldorf, Germany.

and vary over time, and stylized price impact parameters capturing the interaction with the market when executing trades are difficult to estimate in a reliable and persistent manner. To tackle this challenge, we propose a novel sample-based, in-context operator learning framework, termed ICON-OCnet. Specifically, we adapt the In-Context Operator Networks (ICON) learning methodology proposed in [YLMO23] to (i) learn the *unknown* price impact regime prevailing in a *current* market environment from only a few recently executed sample trades, and then (ii) devise the associated optimal order execution strategy, a neural network optimal control policy (called OCnet), for the detected price impact environment.

We examine our general approach within the broad class of linear propagator models proposed by [BGPW04, BFL09, Gat10], where we can effectively benchmark our methodology against the ground truth optimal execution strategies recently obtained in [AJN25]. Propagator models constitute a versatile class of transient price impact models defined by a price impact kernel (propagator), which captures, in reduced form, the interplay between price moves and current and past trades as empirically observed when executing market orders in limit order books. Note, however, that the method developed in this paper does not depend on this particular model setup and can be applied to any price impact framework.

Our proposed ICON-OCnet method is based on an *in-context learning paradigm*, in other words, (transfer-)learning from prompted time series example data serving as context. To this end, we leverage a transformer-based architecture, which is very well suited for sequential data. The idea is to infer the price impact operator, which maps the trading rate as a function of time to the incurred price impact path, using only a few observed sample trajectories. This allows the transformer to adjust to new or unseen price impact models without requiring full retraining of the network. First, we train the ICON model offline on synthetically generated trade and price impact paths from a wide range of propagator model specifications. The pretrained ICON model is then initialized with only a few examples of trades and corresponding realized price impact to infer the underlying price impact operator in a few-shot online learning manner from the prompted context. In a second step, we feed the initialized ICON model as a surrogate operator into the optimal order execution problem and train a neural network policy via a policy gradient method akin to [HE16] to learn the optimal execution strategy for the price impact operator inferred by ICON from the provided context. We validate the performance of our ICON-OCnet approach through various numerical experiments, demonstrating its ability to (i) detect the true price impact operator even when trained on data from a different propagator model class, and (ii) correctly recover the corresponding optimal order execution strategy from the propagator model generating the in-context examples. In particular, the ICON surrogate operator demonstrates sufficient precision and robustness to be effectively utilized in an iterative stochastic gradient descent-type optimization algorithm to learn the corresponding optimal policy.

Finally, the broader purpose of this paper is also to demonstrate a *proof of concept* for our general methodology in a complex and non-trivial setting: solving a path-dependent optimal control problem with unknown state dynamics, inferred in a data-efficient and robust manner from a limited number of in-context examples by leveraging the few-shot and transfer learning capacities of transformer networks.

Related to our work, in the sense of addressing a similar problem but model-specific for linear propagator price impact models, are the studies [NZ23] and [NSZ23]: [NZ23] introduces a statistical online learning approach for linear propagator models, alternating between exploration and exploitation phases, and achieving sublinear regret with high probability; [NSZ23] proposes an offline learning framework to estimate the price impact kernel while accounting for uncertainty in the estimator and derives the asymptotic optimality of strategies in terms of execution cost. In contrast, we suggest in-context, few-shot learning, which efficiently combines offline and online learning, enhancing model flexibility while limiting the need for costly online training. Moreover, our approach is fully model-agnostic and only assumes that the price impact is a function (operator) of the trading trajectory. In particular, our setup is not confined to linear propagator models, and we do not perform any model parameter estimation or classical kernel estimation as in [NSZ23, NZ23]. Instead, we leverage the capabilities of the transformer-based neural network architecture in ICON to detect commonalities in the prompted in-context examples of trade and price impact trajectories with the sample paths encountered during offline training to perform a reliable and stable price impact prediction of executed trades that can be successfully employed in solving a downstream optimal execution problem. To the best of our knowledge, the only work that uses transformer models for trade execution is [KKSH24], which develops an adaptive dual-level reinforcement learning framework based on a combined transformer and Long-Short-Term Memory (LSTM) architecture to track the Volume-Weighted Average Price (VWAP). There has recently been also a lot of interest in the general learning of solution operators for ordinary and partial differential equations with varying network architectures. Among many others, we mention approaches based on Deep Operator Networks (DeepONet) in [LJP+21], graph kernels in [AAB+19], Fourier Neural Networks in [KLL+22], and structure-informed operator learning in [BDG24a, BDG24b].

The rest of the paper is organized as follows: Section 2 introduces the general optimal order execution problem. Our ICON-OCnet approach is described in Section 3. Numerical results are presented in Section 4. Conclusion and outlook are summarized in Section 5.

2 Optimal order execution problem

In this section, we summarize the classical optimal order execution problem in the presence of price impact within a unifying and fairly generic modeling framework that is commonly used in the literature, encompassing various introduced price impact models. We refer to [CJP15, BBDG18, Web23] for general reference.

2.1 Model setup

Let $T > 0$ denote a finite deterministic time horizon and fix a filtered probability space $(\Omega, \mathcal{F}, \mathbb{F} := (\mathcal{F}_t)_{0\leq t\leq T}, \mathbb{P})$ satisfying the usual conditions. We consider a trader with an initial inventory $x \in \mathbb{R}$ in a risky asset (e.g., number of shares held in a stock) who wishes to liquidate her position by time T, which represents, for instance, the end of the trading day. The trader's inventory, i.e., the number of shares held at any time $t \in [0, T]$, is modeled by the process $X = (X_t)_{0\leq t\leq T}$ with dynamics

$$X_t = x - \int_0^t u_s \, ds \qquad (0 \leq t \leq T), \tag{1}$$

where $u = (u_t)_{0\leq t\leq T}$ denotes her selling rate chosen from the set of admissible strategies

$$\mathcal{U} := \left\{ u : \mathbb{F}\text{-progressively measurable s.t. } \int_0^T \mathbb{E}[u_t^2] \, dt < \infty \right\}. \tag{2}$$

We assume that the trader's trading activity causes price impact in the sense that her trades at time $t \in [0, T]$ are filled at the execution price

$$P_t = S_t - Y_t \qquad (0 \leq t \leq T). \tag{3}$$

Here, $S = (S_t)_{0\leq t\leq T}$ denotes a square integrable special semimartingale and represents the unaffected price process, i.e., the price process that would have prevailed without the trader's trading, modeling price changes caused by other traders and the arrival of new information. In contrast, the process $Y = (Y_t)_{0\leq t\leq T}$ models the *price impact* incurred by the trader due to her trading strategy u such that Y_t depends on her current and past order flow $(u_s)_{0\leq s\leq t}$ for all $t \in [0,T]$. We assume $Y_0 = 0$ so that $P_0 = S_0$ in (3).[1]

More precisely, we make the following general assumption.

Assumption 2.1. *We assume that there exists a determinstic, non-anticipative price impact operator $\boldsymbol{I}_\theta : L^2([0,T],\mathbb{R}) \to L^2([0,T],\mathbb{R})$ satisfying*

$$(\boldsymbol{I}_\theta(u))_t = (\boldsymbol{I}_\theta(u|_{[0,t]}))_t \qquad (0 \leq t \leq T) \tag{4}$$

for all $u \in \mathcal{U}$, such that the trader's price impact process Y in (3) is given by

$$Y_t = (\boldsymbol{I}_\theta(u))_t \qquad (0 \leq t \leq T). \tag{5}$$

Henceforth, we will also write the trader's execution price P in (3) as

$$P_t = S_t - (\boldsymbol{I}_\theta(u))_t \qquad (0 \leq t \leq T). \tag{6}$$

In particular, we interpret the operator $\boldsymbol{I}_\theta(\cdot)$ as encoding the *price impact environment* in which the trader executes her trades, and we use the subscript θ for convenience to emphasize the dependence on a certain price impact *regime*. Note that we do not use the term model here to emphasize that $\boldsymbol{I}_\theta(\cdot)$ does not necessarily need to represent a parametric price impact model with some parameter vector θ. For now, we deliberately introduce $\boldsymbol{I}_\theta$ without any further specification to highlight the versatility of our approach. Concrete examples will be considered later in our numerical study in Section 4; cf. Example 4.1.

2.2 Optimization problem

The trader's goal is to find an optimal order scheduling strategy $u \in \mathcal{U}$ that maximizes her objective functional

$$\begin{aligned} J(u) :=& \mathbb{E}\left[\int_0^T P_t u_t \, dt - \varepsilon \int_0^T u_t^2 \, dt - \phi \int_0^T X_t^2 \, dt + X_T S_T - \varrho X_T^2\right] \\ =& \mathbb{E}\left[\int_0^T \big(S_t - (\boldsymbol{I}_\theta(u))_t\big) u_t \, dt - \varepsilon \int_0^T u_t^2 \, dt - \phi \int_0^T X_t^2 \, dt + X_T S_T - \varrho X_T^2\right] \end{aligned} \tag{7}$$

for some constants $\varepsilon > 0$, $\phi \geq 0$ and $\varrho \geq 0$; cf., e.g., [LN19, NV22, AJN25] and the references therein. The first integral in (7) represents the trader's gains from her selling strategy u. The second integral with constant $\varepsilon > 0$ describes in reduced form all additional *instantaneous costs*, which are not directly linked to persistent price impact but incurred, for instance, by the bid-ask spread. The constants $\phi \geq 0$ and $\varrho \geq 0$ implement, respectively, a penalty on the running and terminal inventory. In particular, a large value for ϱ virtually enforces the desired liquidation constraint that X_T will be very close to zero. Lastly, the term $X_T S_T$ represents the final asset position's value in terms of the unaffected fundamental price S_T. Observe that $J(u) < \infty$ for any admissible $u \in \mathcal{U}$.

It is also very common and helpful to rewrite the performance functional in (7) as

$$J(u) = \mathbb{E}\left[\int_0^T \big(\alpha_t - (\boldsymbol{I}_\theta(u))_t\big) u_t \, dt - \varepsilon \int_0^T u_t^2 \, dt - \phi \int_0^T X_t^2 \, dt - \varrho X_T^2\right] + x\mathbb{E}[S_T] \tag{8}$$

[1] It is also conceivable to allow for a general $Y_0 = y \in \mathbb{R}$. In this case, the starting value y models an initial dislocation of P_0 from the fundamental value S_0 at the opening of the continuous trading session on $[0,T]$.

to emphasize the tradeoff of the execution problem between exploiting the asset's short term *intraday alpha signal*

$$\alpha_t := \mathbb{E}[S_t - S_T | \mathcal{F}_t] \qquad (0 \le t \le T) \tag{9}$$

and the incurred price impact $(\boldsymbol{I}_\theta(u))_t$ from liquidation; see [Web23, Chapter 2]. This signal predicts the expected future returns of the unaffected price S during the trading period $[0,T]$. For the optimal execution problem, it serves as a sufficient statistic of the unaffected price process S. Hence, the predictor $\alpha = (\alpha_t)_{0 \le t \le T}$ is usually considered as the relevant additional input variable to the execution problem which is modeled by the controller (trader), rather than S itself; see also the discussion in [HMMKW25, Section 3.2].

To summarize, the aim of the trader is to solve the optimal stochastic control problem

$$J(u) \to \max_{u \in \mathcal{U}}. \tag{10}$$

Observe that the objective function in (7) or, equivalently, in (8) is fairly generic and does not hinge on a specific price impact model. Indeed, the operator $\boldsymbol{I}_\theta(\cdot)$ therein can be any price impact operator. Generally speaking, our goal in the remainder of this paper is to develop a methodology that numerically computes the optimal strategy in (10) when the price impact operator $\boldsymbol{I}_\theta(\cdot)$ in (8) is an *unknown* "black-box" operator that is inferred only from a few trading examples.

Remark 2.2. *We emphasize that in the literature on optimal order execution, it is very common to set up a price impact model by postulating a decomposition of the execution price P as in* (3)*, under the assumption that the unaffected price process S is known and, as a consequence, the impact process Y is observable. Of course, on real financial markets, only the affected execution price P is observed, but not S or Y. As we will explain below, our methodology can be employed when only the observed execution price P is available.*

Remark 2.3. *Note that, due to the representation of the performance functional in* (8)*, the optimal liquidation strategy depends on the intraday alpha signal $(\alpha_t)_{0 \le t \le T}$ in* (9)*, which is very sensible. The process α is the only source of randomness in the objective that needs to be modeled by the trader. Moreover, if the unaffected price process S in* (3) *is a martingale, we have $\alpha \equiv 0$ in* (9)*, and the optimization problem in* (10) *reduces to a deterministic control problem. In this case, the trader does not have any directional view on the unaffected price process S, and the optimal order scheduling strategy is a deterministic function of time $t \in [0,T]$, depending only on the price impact operator $\boldsymbol{I}_\theta$, as well as the hyperparameters $\varepsilon, \phi, \varrho$, and initial inventory x.*

3 Price impact operator learning and neural network solver

In this section, we describe the in-context price impact learning task and downstream optimization algorithm used to find the optimal liquidation strategy in (10) with unknown price impact operator $\boldsymbol{I}_\theta$. An illustration of the method, which we coin ICON-OCnet, is shown in Figure 1. It consists of two steps summarized as follows:

1. Train ICON offline in a data-driven way on a range of price impact models. Then, use the pre-trained ICON model in a few-shot online learning manner to infer the operator $\boldsymbol{I}_{\tilde{\theta}}$ mapping u onto Y in (5) based on only a few examples stemming from a specific model $\tilde{\theta}$, possibly not seen during offline training. Regarding the time series data used as *in-context examples*, there are two possible approaches: one can either use discretized trajectories of pairs (u, Y) or directly prompt discretized realizations of (u, P). For the

former, since Y is not directly observable in practice, it is assumed that the price impact Y is filtered out from the (observed) execution price P in (3) in a pre-processing step.[2]

2. Train a neural network policy (OCnet) via a policy gradient method to approximate the optimal execution strategy in (10) based on the ICON surrogate operator $\hat{\boldsymbol{I}}_{\tilde{\theta}}$ obtained in step 1. In particular, the performance measure for training OCnet is a discretization of the objective functional in (8) with $(\boldsymbol{I}_{\tilde{\theta}}(u))_t$ replaced by the in-context prediction $(\hat{\boldsymbol{I}}_{\tilde{\theta}}(u))_t$ of ICON.

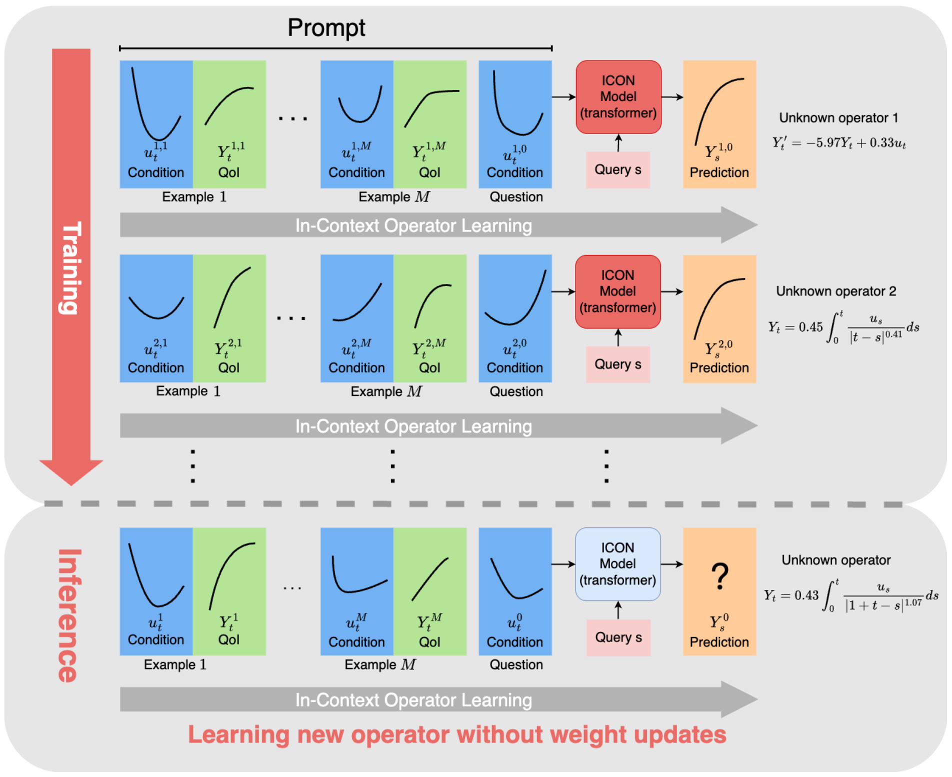
To wit, by prompting only a few samples of time series data of selling rates and corresponding incurred price impact trajectories, our method derives the optimal execution strategy for the unknown price impact environment that generates the examples.

3.1 ICON training and few-shot learning

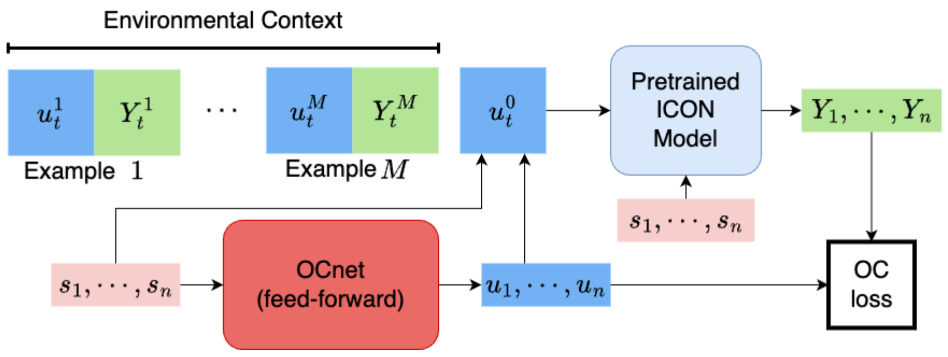
Concerning step 1, inspired by [YLMO23], we use In-Context Operator Networks (ICON), a novel transformer-based neural network architecture designed to learn the operator $\boldsymbol{I}_\theta : L^2([0,T],\mathbb{R}) \to L^2([0,T],\mathbb{R})$ defined in (5) that maps the trading (selling) rate process u to the price impact process $Y = \boldsymbol{I}_\theta(u)$. The network is pre-trained in an offline learning step and can then, in a second online learning step, detect the price impact operator from a few provided examples. Unlike traditional neural network methods that require re-training or fine-tuning for each new problem, the so-called few-shot learning or prompting technique used here requires only a few example trades and their respective price impact in order to recognize the underlying relationship between u and Y. In particular, it can deal with new, unseen price impact models without actually retraining the network weights. This allows the network to handle a wide range of price impact models by leveraging commonalities shared between different models. From a practical perspective, this methodology has several advantages. Specifically, the true relationship $u \mapsto Y$ is typically unknown. Even after a decision on a specific model has been made, there might be uncertainty about the parameters of the model. Moreover, the model parameters might not be constant over time. With the in-context few-shot learning approach, a trading desk tasked with liquidating a large position can, in principle, base its search for an optimal execution strategy in a data-driven way on *experience* from recent trades of the same asset. In this way, the network architecture automatically detects, from a universe of possible price impact models, a model that best fits the most recently *incurred* price impact as provided by the few examples or test trades as context.

Using a transformer architecture also offers multiple benefits for our few-shot prompting task in the context of optimal liquidation. First, its attention mechanism allows the neural network to efficiently process inputs of varying lengths, making it well suited for handling different types of examples (data prompts) and key-value pairs (i.e., time instances s_j and observations (u_{s_j}, Y_{s_j}), respectively (u_{s_j}, P_{s_j})) without necessitating structural changes. More precisely, the transformer accepts inputs up to a maximum sequence length, with shorter sequences padded and masked accordingly. From a practical point of view, this flexibility is very helpful in learning the true price impact operator. Indeed, the trades and the corresponding observed price impact or execution prices that are used as examples are usually not uniform in length and are sampled at different, possibly non-equidistant and fairly irregular time steps. Second, the dependence of the output on the input of transformers can be quite flexible, and hard constraints can be added through the design of specific masks. For instance,

[2]We refer to the related discussion of fitting price impact models in [Web23, Chapter 7] and the references therein.



(a) Step 1: ICON training



(b) Step 2: OCnet training

Figure 1: Illustration of the ICON-OCnet structure. In both steps, the red rounded rectangle represents the training of a neural network, while the blue rounded rectangle indicates a pre-trained neural network with frozen parameters.

our relationship between u and Y is non-anticipative; hence, we use a modified causal mask to add this constraint. Moreover, and most importantly for our purposes, the multi-head attention mechanism of the transformer is very well suited for *sequential time series data* and enables capturing the relationships between different parts of the input. This facilitates the learning of the price impact by focusing on the most relevant aspects of the examples. As a consequence, transformers can deal with complex interdependencies, making them a very natural choice for a typically *non-Markovian* price impact environment, which relates price moves intertemporally to current and past trades. Lastly, transformers also support parallel processing, rendering it computationally efficient to generate predictions across multiple query points simultaneously, which is vital for fast inference in few-shot settings. We also refer to the discussion in [YLMO23].

3.1.1 ICON training with in-context pairs (u, Y)

We first explain the training procedure and use-case for an idealized situation where the trading-incurred price impact Y is perfectly known and available as context for offline training and online inference. For the sake of clarity, this will also be the setup in our proof-of-concept experiment with synthetic time series data in Section 4 below. The corresponding in-context learning procedure and few-shot prompting inference of the ICON model are illustrated in Figure 1 (a).

First, each row above the dashed line represents one data point that is used in training. The data in row i is produced by the same price impact model θ_i and consists of M "condition" and "quantity of interest (QoI)" pairs of discretized trajectories $(u^{i,1}, Y^{i,1}), \dots, (u^{i,M}, Y^{i,M})$, observed on possibly irregular time grids $t_1^i, \dots, t_{l^i}^i$, serving as the context, followed by the "question" (or "question condition") $u^{i,0}$ specified on some time grid $s_1, \dots, s_k$, the associated "query" $(s_1, \dots, s_k)$, and the corresponding "prediction" $(Y_{s_1}^{i,0}, \dots, Y_{s_k}^{i,0})$. The query provides ICON with the grid information on which the prediction is evaluated. Note that the discretization for the question condition $(s_1, \dots, s_k)$ can differ from the discretization of the example pairs $(t_1^i, \dots, t_{l^i}^i)$, $i \in \{1, \dots, M\}$, which may even vary across the M examples themselves. Based on this data, ICON is trained in an offline manner on different models. In Figure 1 (a), this is illustrated by the different price impact operators of different linear propagator models, which we use in our numerical study in Section 4. We use a suitable self-attention mechanism via masking to ensure that Y is learned and predicted in a non-anticipative way with respect to u (in the provided examples and in the prediction for the question condition); see also the discussion below.

Next, at the inference stage illustrated below the dashed line in Figure 1 (a), the trained ICON network is presented with M new time series sample data (data prompts) of (u, Y) pairs, stemming from a possibly unseen model $\tilde{\theta}$, discretized on some time grid. Based on these examples, the pre-trained ICON network infers the underlying price impact operator $\boldsymbol{I}_{\tilde{\theta}}$ and provides the output prediction for an arbitrary question condition u^0 at an arbitrary query $(s_1, \dots, s_n)$. In other words, it predicts the realized price impact $Y^0 = \boldsymbol{I}_{\tilde{\theta}}(u^0)$ incurred by a strategy u^0 on the time grid $(s_1, \dots, s_n)$ for a *possibly unknown model* $\tilde{\theta}$ that generated the sample time series data as context. Moreover, predictions of the price impact $\boldsymbol{I}_{\tilde{\theta}}(u)$ for any strategy $u \in \mathcal{U}$ on a varying time grid can then be obtained by providing as context the same M example data pairs from the underlying model $\tilde{\theta}$ in every inference step. In this way, we obtain an ICON *surrogate operator* $\hat{\boldsymbol{I}}_{\tilde{\theta}}$ for the true operator $\boldsymbol{I}_{\tilde{\theta}}$ that we can feed into an optimization algorithm to compute the corresponding optimal order execution strategy in (10). The subscript $\tilde{\theta}$ in the notation for the surrogate operator $\hat{\boldsymbol{I}}_{\tilde{\theta}}$ emphasizes that the provided context (examples) originates from the same model $\tilde{\theta}$. We stress, however, that ICON does not actually infer the underlying model $\tilde{\theta}$ and its parameters. Instead, it internally processes an encoder-based representation of the model underlying the examples, which is based on similarities with the models generating the examples in the offline training step. In particular, the model $\tilde{\theta}$ could be a completely new, unseen model that is even outside the class of models encountered in training, as long as this family of models is rich enough to embed important characteristics of $\tilde{\theta}$.

Our ICON neural network architecture corresponds to the one used in [YLO25] for learning solution operators of partial differential equations. To implement the non-anticipative structure within the transformer, we construct a causal (self-)attention mask that enforces the correct temporal dependency between the control u and its response Y, in the provided examples serving as context as well as in the actual question condition. That is, the mask ensures that, when learning/predicting a value Y_{s_i} at some time instant s_i, the model can only self-attend to the past control inputs u_{s_j} at instances $j \leq i$, thereby preserving the causal relation between the trajectories u and Y, and preventing any information leakage from the future. Moreover, multiple example pairs are arranged within one sequence so that the model simultaneously learns from several input-output relations. This sequence design allows earlier examples to also serve as context for later examples, and hence enables the transformer to learn consistent causal dependencies across varying temporal grids. We present the attention mask in a simplified setup for the online inference step (i.e., step 1 in Figure 1 (a) below the dashed line) in Figure 2, where the yellow regions indicate allowed self-attention and the dark regions indicate masked connections. The block lower-triangular structure reflects the causal, non-anticipative nature of the inference process, as well as the fact that earlier examples also serve as context for later examples as mentioned above; see also the caption of Figure 2 for more details. The mask we use for training ICON (i.e., for step 1 in Figure 1 (a) above the dashed line) is more involved and not presented here. More details on the training are provided in Sections 4.2 and 4.3.

3.1.2 ICON training with in-context pairs (u, P)

We now outline the changes to the offline and online training procedures when time series data of pairs (u, P) of trading strategies u and corresponding realized execution price trajectories P in (3) are to be used as the actual in-context examples. In this case, ICON can, in principle, be trained to filter out the price impact from the context. Note, however, that in contrast to Y, the realizations of P are noisy due to the additional presence of the unaffected price process S.

More precisely, as long as the offline training step is model-based, using synthetic time series data of the execution price P in (3) with known trading-incurred price impact Y as in Section 3.1.1 above (and a known model for S), one can generate a training set that consists of (noisy) realizations $(u^{i,1}, P^{i,1}), \ldots, (u^{i,M}, P^{i,M})$ serving as the M context examples, followed by the question condition $u^{i,0}$, a query $(s_1, \ldots, s_k)$, and the prediction $(Y^{i,0}_{s_1}, \ldots, Y^{i,0}_{s_k})$ of the true price impact. Here, $Y^{i,0}$ is consistent with the model generating the in-context execution prices $(P^{i,1}, \ldots, P^{i,M})$. Based on this data, ICON can be trained offline on a wide range of different price impact models as described above and illustrated in Figure 1 (a), except that the in-context example pairs $(u^{i,1}, Y^{i,1}), \ldots, (u^{i,M}, Y^{i,M})$ are now replaced by realizations $(u^{i,1}, P^{i,1}), \ldots, (u^{i,M}, P^{i,M})$. For the online few-shot inference step, the trained ICON network is then presented with M new example trajectories $(u^1, P^1), \ldots, (u^M, P^M)$ and predicts the realized price impact $Y^0 = \boldsymbol{I}_{\tilde{\theta}}(u^0)$ of an arbitrary prompted strategy u^0 on some query $(s_1, \ldots, s_n)$, based on the detected impact model $\tilde{\theta}$ underlying the in-context examples, by leveraging commonalities with the learned models from the offline training.[3]

[3]In practice, proprietary trading data could serve as prompted examples, each consisting of a meta-order execution strategy that was executed over some time period via a sequence of smaller child orders.

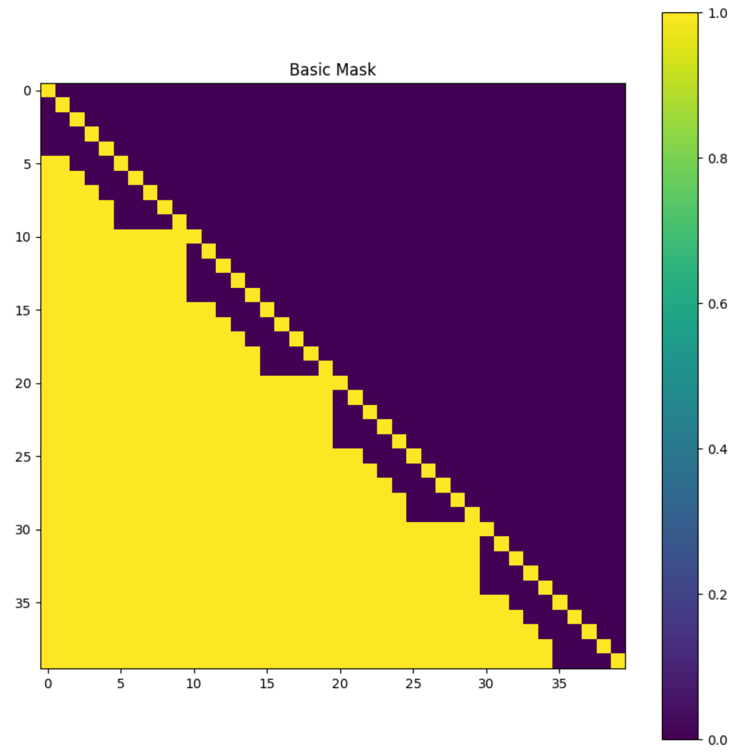


Figure 2: Representation of the attention mask used in ICON during the inference step in a simplified setup with only $M = 3$ example pairs $(u^m, Y^m)_{m=1,\dots,3}$ and a fixed time grid $s_0, s_1, \dots, s_4$ with $s_0 = 0$. More precisely, in all three examples, the condition consists of $(Y^m_{s_0}, u^m_{s_0}, u^m_{s_1}, u^m_{s_2}, u^m_{s_3})$ and the quantity of interest is given by $(Y^m_{s_0}, Y^m_{s_1}, Y^m_{s_2}, Y^m_{s_3}, Y^m_{s_4})$. Note that we effectively include the starting value Y^m_0 as the first value in the condition (for the sake of generality, even though we assume $Y_0 = 0$ throughout) and in the QoI. The desired prediction is then $Y^0_{s_0}, Y^0_{s_1}, Y^0_{s_2}, Y^0_{s_3}, Y^0_{s_4}$ with query $(s_0, s_1, s_2, s_3, s_4)$ for a given question condition $(Y^0_{s_0}, u^0_{s_0}, u^0_{s_1}, u^0_{s_2}, u^0_{s_3})$. The indices of the input sequence are arranged on the vertical axis from 0 to 39, and from the horizontal axis one can read off the indices it "self-attends" to (yellow regions). More precisely, the input is arranged as one sequence, starting with the three examples $(Y^1_{s_0}, u^1_{s_0}, \dots, u^1_{s_3}, Y^1_{s_0}, \dots, Y^1_{s_4}, \dots, Y^3_{s_0}, u^3_{s_0}, \dots, u^3_{s_3}, Y^3_{s_0}, \dots, Y^3_{s_4})$ (indices 0-29), followed by the question condition $Y^0_{s_0}, u^0_{s_0}, \dots, u^0_{s_3}$ (indices 30-34) and query times $s_0, \dots, s_4$ (indices 35-39). From the graph we see that the trained model self-attends the value $Y^m_{s_i}$ in the m-th example only to past values $u^m_{s_j}$ with $j \le i$ and the starting value $Y^m_{s_0}$, without access to future information (neither in u^m nor Y^m); as well as to all previous example pair values $(Y^n_{s_0}, u^n_{s_0}, \dots, u^n_{s_3}, Y^n_{s_0}, \dots, Y^n_{s_4})$, $n < m$, on the entire grid, serving as additional context for the m-th example. The same logic applies for the actual prediction of the value $Y^0_{s_i}$ for the query input s_i, $i \in \{0, \dots, 4\}$, in the last five rows (input indices 35-39), where only the values $Y^0_{s_0}$, $u^0_{s_j}$, $j \le i$, in the input are attended (from the indices 30-34), as well as all previous example pair values $(u^m_{s_l}, Y^m_{s_l})_{m=1,\dots,3}$ (indices 0-29) serving as the context.

Generally speaking, the offline training and online inference procedure of ICON can be seen as being conditional on the underlying model in the prompted examples. When offline training is based on in-context examples $(u^{i,1}, Y^{i,1}), \ldots, (u^{i,M}, Y^{i,M})$ as outlined in Section 3.1.1 above, this conditional training and inference is a classical operator learning problem for the deterministic operator $\boldsymbol{I}_{\tilde{\theta}} : L^2([0,T]) \to L^2([0,T]), u \mapsto \boldsymbol{I}_{\tilde{\theta}}(u)$ in (5), which is performed very efficiently by facilitating transfer learning from other models encountered during the offline training phase. In contrast, when training is done based on noisy in-context examples $(u^{i,1}, P^{i,1}), \ldots, (u^{i,M}, P^{i,M})$, the offline learning and online few-shot inference step is no longer a classical operator learning problem. Instead, the object of inference is a map $\boldsymbol{P}_{\tilde{\theta}} : L^2([0,T]) \to \mathcal{M}(L^2([0,T])), u \mapsto \boldsymbol{P}_{\tilde{\theta}}(u)$, where $\mathcal{M}(L^2([0,T]))$ is the set of probability measures on $L^2([0,T])$ and $\boldsymbol{P}_{\tilde{\theta}}(u)$ denotes the distribution of the trajectory of the execution price P in (3) under model $\tilde{\theta}$ when trading follows strategy u. For a given question condition u^0, this map is then queried for the expectation $Y^0_{\cdot} = \mathbb{E}_{\boldsymbol{P}_{\tilde{\theta}}(u^0)}[P_0 - P_{\cdot}]$ in (3) at points $(s_1, \ldots, s_k)$ (assuming that the unaffected price process S in (3) is a martingale), where the expectation is to be understood in a functional sense, i.e., as an element in $L^2([0,T])$. If S is not a martingale, then $Y^0_{\cdot} \neq \mathbb{E}_{\boldsymbol{P}_{\tilde{\theta}}(u^0)}[P_0 - P_{\cdot}]$, and a shifted expectation is learned instead. In general, it is expected that when ICON is trained based on noisy in-context pairs $(u^{i,1}, P^{i,1}), \ldots, (u^{i,M}, P^{i,M})$, the training set and the number of examples M must be considerably larger than when ICON is trained directly based on examples $(u^{i,1}, Y^{i,1}), \ldots, (u^{i,M}, Y^{i,M})$ as in Section 3.1.1.

3.2 Neural network solver with ICON surrogate

We now explain the general methodology for step 2, that is, the OCnet training for the optimal liquidation strategy using ICON as a proxy for the price impact operator. This step is illustrated in Figure 1 (b). More precisely: (i) as described above in Section 3.1.1, based on M discretized example trajectories $(u^i, Y^i)_{i=1,\ldots,M}$ provided as context, ICON infers a model (here represented by θ) and we obtain an ICON surrogate operator $\hat{\boldsymbol{I}}_\theta$, which approximates the true price impact operator $\boldsymbol{I}_\theta$ in (5); (ii) we feed $\hat{\boldsymbol{I}}_\theta$ into the optimization problem in (10) and train a neural network policy (OCnet) to approximate the optimal execution strategy in the presence of the current price impact environment detected by ICON.

Put differently, for a fixed initial inventory $x \in \mathbb{R}$, using the representation in (8), the optimization problem becomes

$$\max_{u \in \mathcal{U}} \mathbb{E}\left[\int_0^T \Big(\big(\alpha_t - (\hat{\boldsymbol{I}}_\theta(u))_t\big)u_t - \varepsilon u_t^2 - \phi X_t^2\Big)dt - \varrho X_T^2\right], \tag{11}$$

where the in-context transformer network predicts the price impact

$$Y_t \approx \hat{\boldsymbol{I}}_\theta\left(t, (u_s)_{0\le s\le t}; (u^1_s, Y^1_s)_{0\le s\le T}, \ldots, (u^M_s, Y^M_s)_{0\le s\le T}\right) \qquad (0 \le t \le T) \tag{12}$$

incurred by a strategy $u \in \mathcal{U}$. With a slight abuse of notation in (12), we make explicit the path dependence of Y_t on all past trades $(u_s)_{0\le s\le t}$, as well as the dependence on the M fixed example pairs observed on $[0,T]$ provided as context.

Next, assuming that trading takes place on an equidistant time grid $t_i = i\Delta t$, $i = 0, \ldots, N-1$, with $\Delta t = T/N$, and having access to L samples $(\alpha^{(l)}_{t_i})_{i=0,\ldots,N-1}$, $1 \le l \le L$, of the alpha signal $(\alpha_t)_{0\le t\le T}$ in (9) (modeled by the trader), one can approximate (11) via

$$\frac{1}{L}\sum_{l=1}^{L}\Bigg\{\sum_{i=0}^{N-1}\bigg(\Big(\alpha_{t_i}^{(l)}-\hat{\boldsymbol{I}}_{\theta}\big(t_i,u_{t_0},\dots,u_{t_{i-1}};(u_{t_j}^{1},Y_{t_j}^{1})_{j=0,\dots,N-1},\dots,(u_{t_j}^{M},Y_{t_j}^{M})_{j=0,\dots,N-1}\big)\Big)u_{t_i} \\ -\varepsilon u_{t_i}^2-\phi X_{t_i}^2\bigg)\Delta t-\varrho X_{t_N}^2\Bigg\}\to\max_{u\in\mathcal{N}} \tag{13}$$

with $X_{t_i}=x-\sum_{j=0}^{i-1}u_{t_j}\Delta t$. Here, $\mathcal{N}$ denotes the set of all feed-forward neural network policies of the form $\mathrm{NN}_\vartheta(t,a)$ with $\mathrm{NN}_\vartheta:[0,T]\times\mathbb{R}\to\mathbb{R}$ and some fixed architecture parameterized by $\vartheta\in\mathbb{R}^m$ for some $m\in\mathbb{N}$. Specifically, Remark 2.3 motivates in (13) to search for neural network policies $\mathrm{NN}_\vartheta\in\mathcal{N}$ as functions in $(t_i,\alpha_{t_i}^{(l)})$, i.e., $u_{t_i}=\mathrm{NN}_\vartheta(t_i,\alpha_{t_i}^{(l)})$ and realized alpha signal $\alpha^{(l)}$. The optimal strategy can then be readily computed by employing a policy gradient method; that is, by adopting the approach in [HE16], running a stochastic gradient descent-type algorithm on the discrete objective functional in (13) with respect to the neural network parameters ϑ. We refer to the obtained optimal execution strategy as the OCnet policy. More details on the OCnet training are provided in Section 4.4.

4 Numerical illustrations

In this section, we describe our numerical experiments and the results obtained. Specifically, we study our general approach within the broad class of linear propagator models. This class of models was originally developed by [BGPW04, BFL09] in discrete time and formulated by [Gat10] in continuous time; see also [BBDG18, Chapter 13]. The associated optimal liquidation problem was explicitly solved only very recently in [AJN25]. We emphasize that the methodology proposed in this paper does not depend on this particular model setup but can be applied to any price impact model specification. We merely focus our numerical study on the linear propagator models because their tractability readily allows us to effectively benchmark our method against explicit optimal solutions (ground truths).

Example 4.1. *In the class of linear propagator models, the price impact process* $Y=(Y_t)_{0\le t\le T}$ *in* (5) *is modeled by an integral operator of the form*

$$Y_t=(\boldsymbol{I}_\theta(u))_t=\int_0^T G(t-s)1_{\{s\le t\}}u_s\lambda\,ds \qquad (0\le t\le T) \tag{14}$$

with some price impact kernel $G:[0,T]\to\mathbb{R}_+$ *in* $L^2([0,T],\mathbb{R})$ *(also called propagator) and constant push factor* $\lambda>0$ *(also referred to as Kyle's lambda). In particular, the process* Y *represents transient price impact that persists and decays over time. The types of parametric kernels* G *we are considering in* (14) *are as follows:*

(I) exponential kernels as proposed by [OW13] of the form

$$G(t)=e^{-\beta t} \qquad (0\le t\le T) \tag{15}$$

with some impact decay parameter $\beta>0$*. In this case, note that* Y *in* (14) *is a Markovian process and satisfies the (random) linear ordinary differential equation (ODE)*

$$Y_0=0, \qquad \dot{Y}_t=-\beta Y_t+\lambda u_t \qquad (0\le t\le T). \tag{16}$$

(II) power law kernels as introduced by [BGPW04, Gat10] of the from

$$G(t)=\frac{1}{(\ell+t)^\gamma} \qquad (0\le t\le T) \tag{17}$$

for some shift parameter $\ell \geq 0$ and decay elasticity $\gamma > 0$. We distinguish between (i) the non-singular case with $\ell > 0$ and $\gamma > 0$, and (ii) the singular case with $\ell = 0$ and $\gamma \in (0, 0.5)$. Observe that power law kernels induce non-Markovian dynamics for the impact process Y in (14).

Here, we use the subscript θ in $\boldsymbol{I}_\theta$ in (14) to emphasize the dependence of the operator on a certain price impact propagator model with a parametric kernel G of either type (I) or (II) and a push factor λ.

Using the class of propagator models from Example 4.1 as a testbed, we illustrate the following:

1. The trained ICON model is capable of accurately inferring the underlying price impact model from only $M = 5$ examples provided as context, even for propagator models not present in the training data.

2. Using ICON as a surrogate operator in the optimal execution problem (ICON-OCnet) correctly retrieves the ground truth optimal execution strategies $u^\star$ from [AJN25, Proposition 4.5] for various propagator models generating the five examples as context.

Throughout our numerical study, we set $T = 1$ and perform all training steps for ICON and OCnet, including the in-context examples and queries for the predictions, on a fixed equidistant time grid with $N = 100$ steps. Note that this is merely for simplicity and not a requirement. In addition, the hyperparameters in the optimal control problem in (13) are set as follows: we fix the instantaneous cost parameter ε at 0.5 and set $\phi = 0$ for the penalty on the running inventory in accordance with the implementation of the ground truth optimal strategies $u^\star$ in [AJN25, Section 5], which we use as our benchmark. We also let the penalization on the terminal inventory ϱ be equal to 10 to enforce the liquidation constraint $X_T \approx 0$.

Moreover, to get a clear picture of the performance of our ICON-OCnet methodology, we focus as in [NSZ23, Section 3] on the deterministic base case as a testbed and "neutralize" the effects of an exogenous intraday alpha signal by assuming that the unaffected price process S in (3) is a martingale. Note that this implies $\alpha \equiv 0$ in (9). Hence, without loss of generality, it suffices to optimize in (13) over neural network policies that are functions in time only; recall also Remark 2.3. In particular, we know from the results in [AJN25] that in this case the optimal selling rates $u^\star$ to liquidate an initial positive inventory $x > 0$ are strictly positive, smooth, U-shaped functions in time with varying curvature depending on the propagator kernel G and the push factor λ. We refer to [AJN25, Section 5.2] for more details.

4.1 Model parameters

For our numerical study, we adopt the parameter configuration used in [NWZ23, Section 3]. The time unit is trading days and is denoted by $[T]$. We have $T = 1$ so that the strategy trades during one trading day over the interval $[0, 1]$ (i.e., 6.5 hours during Nasdaq's opening hours). All trading quantities are expressed as a percentage of the *Average Daily Volume* (ADV%) and we denote by $[V]$ the volume unit. For instance, a sell order with initial size $x = 0.1\,[V]$ represents a sell order of size 10% ADV. Accordingly, the trading rate u in (1) is measured in $[V][T]^{-1}$. We focus on a range of $[0.01, 0.20]$ for the initial inventory x.

The natural (absolute) unit of Y (and the unaffected price process S) in (3) is USD [\$]. Ultimately, however, it is more common to think of the price impact Y (and its associated costs) in terms of basis points (bps) relative to some initial benchmark price (e.g., the prevailing decision price $P_0 = S_0$ at the beginning of the trading period). The performance functional in (8), respectively (13), is then also standardized accordingly to represent total implementation shortfall costs in basis points. Therefore, in the units of the parameters below, we treat Y as unitless (i.e., standardized). For the kernel parameters in (I) and (II), and the push factor λ in Example 4.1, we focus on the following ranges:

(i) The impact decay β in (16) has unit $[T]^{-1}$ and describes the speed at which impact reverts to zero. The corresponding half-life is given by $\log(2)/\beta$. We let β vary in the interval $[0.462, 9.011]$, which roughly corresponds to half-lives ranging from 30 minutes to 1.5 days.

(ii) The push factor λ in (14) has unit $[V]^{-1}$ with values in $[0.1, 0.5]$.

(iii) The power law kernel in (17) captures multiple timescales of decay in a parsimonious way. In the non-singular case, we set $\ell = 1$ and let γ vary in $[0.3, 1.5]$. For the singular case $\ell = 0$, we restrict γ to $[0.35, 0.45]$.

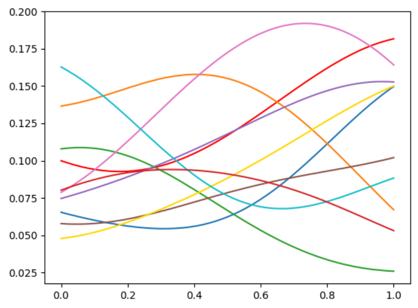
4.2 Data generation

We follow the training procedure outlined in Section 3.1.1 based on in-context pairs (u, Y). For the offline training step of ICON (recall Figure 1 (a)), we generate three different synthetic datasets of selling rates and price impact trajectories (u, Y) corresponding to the three different kernels in (15) and (17): exponential, non-singular power law, singular power law. Here, we take the exponential kernel in (15) as an example to explain our training methodology. First, we randomly sample 80,000 pairs of hyperparameters $\theta = (\lambda, \beta)$ for the push factor $\lambda \sim U([0.1, 0.5])$ and the impact decay $\beta \sim U([0.462, 9.011])$; cf. also Section 4.1. Next, for each θ, we generate 10 strictly positive selling rates $u = \tilde{u} + 0.1$, where $\tilde{u}$ is sampled from a smooth Gaussian process on $[0, 1]$ with kernel $0.05^2 \exp(-2t^2)$. Recall from Section 4.1 that the selling rate is measured in $[V][T]^{-1}$ with ADV as volume unit. We then compute for each u the corresponding price impact $Y = \boldsymbol{I}_\theta(u)$ using a discretized version of (14). The other two datasets for the power law kernels in (17) are generated similarly, with hyperparameters $\lambda \sim U([0.1, 0.5])$, $\gamma \sim U([0.3, 1.5])$ and $\ell = 1$ for the non-singular kernel; and $\lambda \sim U([0.1, 0.5])$, $\gamma \sim U([0.35, 0.45])$ for the singular kernel with $\ell = 0$. An illustration of 10 sample trajectories used in training is shown in Figure 3.

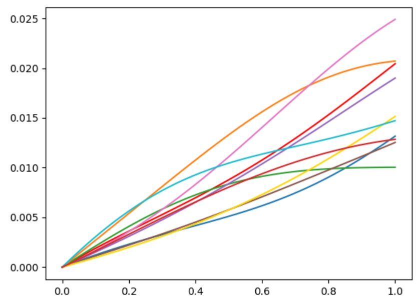
During each iteration in the training process of ICON, a mini-batch of size 8 of hyperparameters θ_i, $i = 1, \ldots, 8$, is randomly selected from the 80,000 generated data points. Then, for each such θ_i with associated 10 generated sample pairs $((u^{i,1}, Y^{i,1}), \ldots, (u^{i,10}, Y^{i,10}))$ as described above, we randomly select one pair as the input-output pair (i.e., question condition $u^{i,0}$ and labeled output $Y^{i,0}$ as illustrated in Figure 1 (a)); and from the remaining 9 pairs we randomly select $M = 5$ to serve as examples in the context. A stochastic gradient descent step (using scheduled AdamW) on the training loss function (i.e., update on the ICON's parameters) is then performed with this mini-batch of 8 data points. The loss function is a standard mean squared error of the prediction on the time grid on $[0, 1]$, referred to as the ℓ^2 error.

4.3 ICON performance

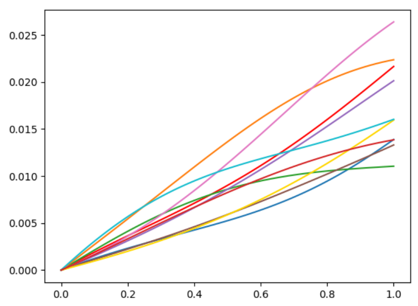
In this section, we evaluate the performance of ICON training described in Section 3.1.1. In total, four different ICON models are trained: (i) three models, each trained on a separate propagator model dataset `ode`, `ker`, `sker` generated from the three different propagator kernels exponential (`ode`), non-singular power law (`ker`), singular power law (`sker`) as described above in Section 4.2; (ii) a fourth model trained on a mixed dataset `all3` that combines the previous three datasets. To ensure a fair comparison, the number of training samples is 80,000 in all



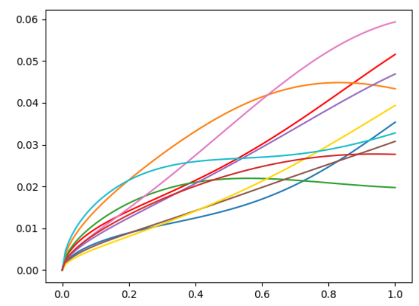
(a) selling rates u



(b) exponential Y



(c) non-singular Y



(d) singular Y

Figure 3: Illustration of 10 training trajectories (selling rates u and corresponding price impact Y) for the three different kernels with parameters $\lambda = 0.2$, $\beta = 0.5$, $\gamma = 0.45$.

four datasets (i.e., in `all3` only 1/3 of each dataset `ode`, `ker`, `sker` of size 80,000 is used). Inspired by [YLO25] we use an encoder-only transformer architecture with 6 layers, 8 heads, head dimension 256, model dimension 256, and a widening factor of 4. We train all four ICON models for 100,000 iterations, roughly corresponding to 10 epochs. Training is performed on a single NVIDIA GeForce RTX 4090 GPU and takes about 3.5 hours. We refer the reader to [VSP+17] where this architecture was first introduced, and [Ala18] for a non-technical illustration. The resulting ℓ^2 relative error is plotted in Figure 4 against the number of iterations.

After training, we assess the relative ℓ^2 errors of all four ICON model predictions $\hat{\boldsymbol{I}}_\theta(u)$ with respect to the exact output $Y = \boldsymbol{I}_\theta(u)$ of the provided question condition u, for various in-context examples from propagator models θ from out-of-sample test datasets. In particular, we examine the out-of-distribution performance (i.e., transfer learning) of the first three ICON models (trained separately on `ode`, `ker`, `sker`, respectively) on in-context examples from a propagator kernel type not seen during the training phase. The test datasets, denoted by `ode_t`, `ker_t`, `sker_t`, are generated using the exact same methodology as for the training data described in Section 4.2. The results are summarized in Table 1. Each column represents a trained ICON model and each row corresponds to a specific test dataset for the in-context examples. For each combination, we calculate the mean and standard deviation of the relative ℓ^2 prediction error for a sample of size 576. The in-distribution errors (bold values in the diagonal of the first three columns) are very small for each ICON model, illustrating that the trained ICON models are capable of accurately inferring the underlying propagator model from the $M = 5$ in-context examples. Moreover, also the out-of-distribution errors for the transfer learning are quite small (off-diagonal entries of the first three columns). Hence, the

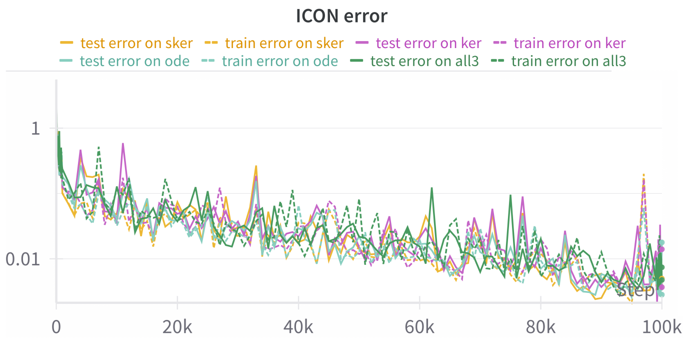


Figure 4: ICON error (on the training set and a separate test set) versus training iterations with 100,000 training steps in total.

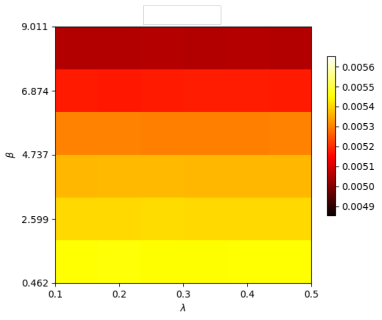
ICON prediction based on few-shot learning from the prompted examples generalizes fairly well to unseen propagator kernel types. We also observe that the ICON model trained on the mixed dataset `all3` performs very well across all three in-context example training sets.

	`ode`	`ker`	`sker`	`all3`
`ode_t`	$\mathbf{0.0053 \pm 0.0045}$	0.1075 ± 0.1266	0.1635 ± 0.2369	0.0060 ± 0.0044
`ker_t`	0.0072 ± 0.0057	$\mathbf{0.0045 \pm 0.0024}$	0.0345 ± 0.0309	0.0063 ± 0.0048
`sker_t`	0.0423 ± 0.0191	0.0392 ± 0.0241	$\mathbf{0.0052 \pm 0.0036}$	0.0057 ± 0.0036

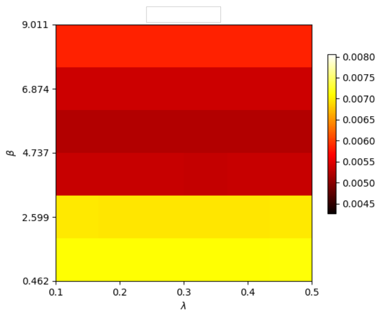
Table 1: ICON error table of relative ℓ^2 prediction errors. Each column corresponds to one of the four ICON models trained on a specific dataset. The rows represent the different test datasets for the 5 in-context examples.

Next, we present in Figure 5 heatmaps which shed some light on the dependence of the ICON models' in-distribution prediction errors with respect to the underlying price impact model parameters generating the in-context examples. More precisely, the two axes represent the hyperparameters θ for each kernel type (I) and (II) (split in 6 intervals), with the color of each box indicating the mean error of the ICON prediction for 16 randomly selected hyperparameters within the box's range which are used for generating the in-context example pairs and the exact output label $\boldsymbol{I}_\theta(u)$ of the question condition u (again, everything is generated using the same method as described in Section 4.2). Note that the error remains invariant with respect to the push factor λ due to the scaling-invariant property of our ICON architecture. With regards to the impact decay β in the exponential kernel, the error becomes smaller with faster decay (Figure 5, (a) and (b)). For the power law kernel, the influence of γ shows a slightly less coherent pattern across the different ICON models `ker`, `sker`, `all3` (Figure 5, (c)–(f)); perhaps not surprisingly due to the more subtle nature of a power law decay. Overall, however, the errors are very small.

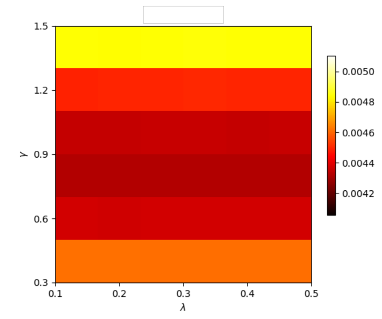
Finally, we evaluate how well ICON generalizes to *unseen* selling rates u as question conditions, that is, strategies which are not directly generated from the Gaussian process as in the synthetic training datasets. This is a first sanity check of ICON's effectiveness as a surrogate operator in the downstream optimal execution problem in (13), i.e., our ICON-OCnet approach. To this end, we generate five synthetic examples as context from a specific propagator model θ using the same method as described in Section 4.2. However, for the in-context prediction task, we use as question condition the actual (out-of-distribution) ground truth optimal execution strategy $u^\star$ of model θ which is computed via the implementation provided by [AJN25]. We then compute the corresponding optimal price impact process



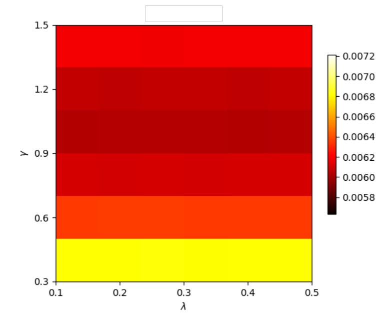
(a) `ode_t`, `ode`



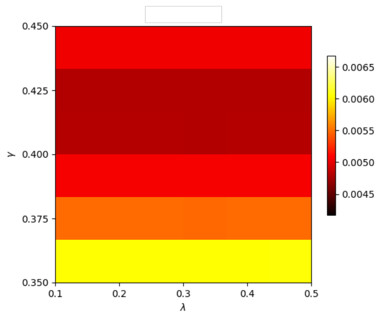
(b) `ode_t`, `all3`



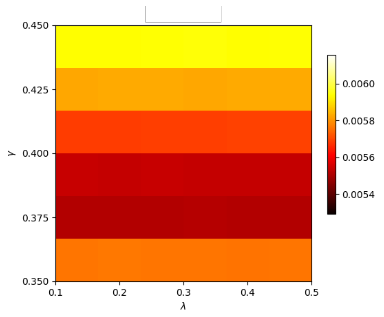
(c) `ker_t`, `ker`



(d) `ker_t`, `all3`

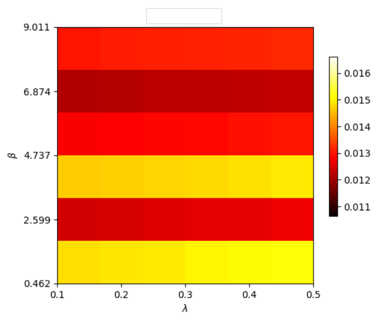


(e) `sker_t`, `sker`

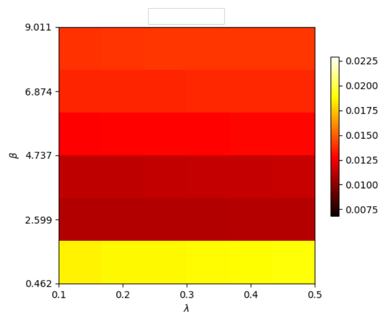


(f) `sker_t`, `all3`

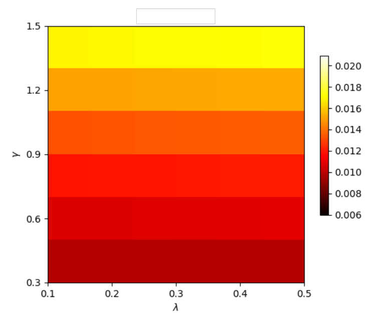
Figure 5: Heatmaps for ICON in-distribution errors for different types of in-context examples (first label) and ICON models trained on a specific dataset (second label). The value of each box represents the mean error over 16 random samples of sets of hyperparameters θ from the corresponding ranges, generating the five in-context examples (x-axis represents values for λ, y-axis represents values for β and γ, respectively).



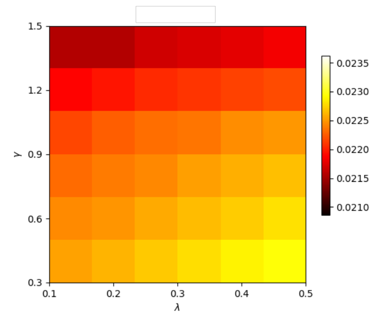
(a) `ode_t`, `ode`



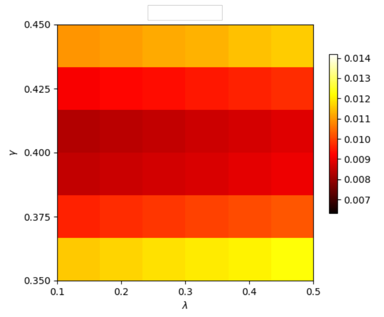
(b) `ode_t`, `all3`



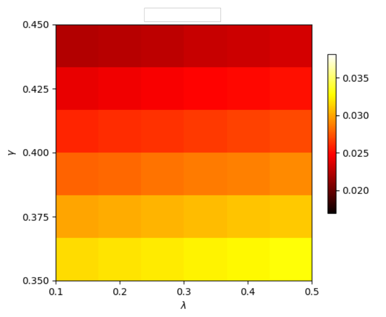
(c) `ker_t`, `ker`



(d) `ker_t`, `all3`

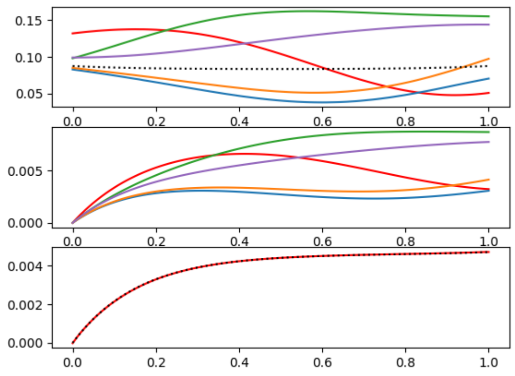


(e) `sker_t`, `sker`

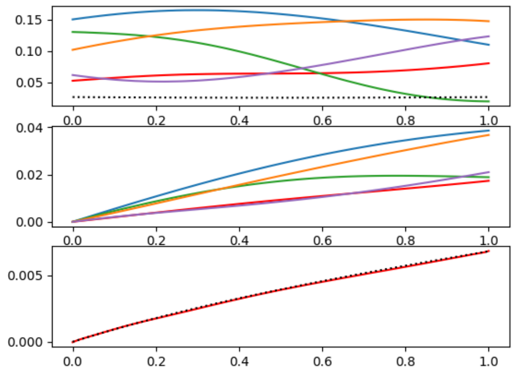


(f) `sker_t`, `all3`

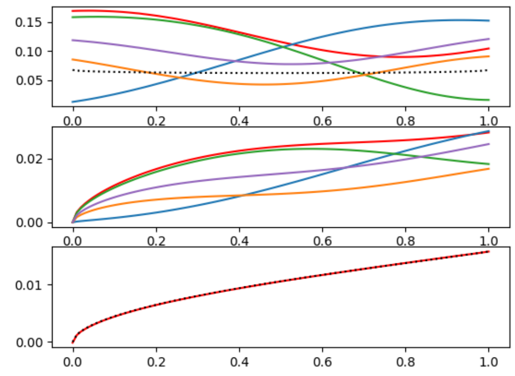
Figure 6: Heatmaps similar to Figure 5 for ICON errors for different types of in-distribution in-context examples (first label) and ICON models trained on a specific dataset (second label). Here, the question condition for the ICON prediction is the out-of-distribution optimal execution strategy $u^\star$ of the corresponding propagator model associated with hyperparameter θ. The value of each box represents the mean error over 16 random samples of sets of hyperparameters θ from the corresponding ranges (x-axis represents values for λ, y-axis represents values for β and γ, respectively).



(a) exponential kernel



(b) non-singular power law kernel



(c) singular power law kernel

Figure 7: Illustration of the out-of-distribution prediction for $i = 1, 2, 3$ different ICON models trained on `ode` (a), `ker` (b), `sker` (c), respectively. *Top panels:* Five selling rates $u^{i,1}, \ldots, u^{i,5}$ as in-context example conditions (colored solid lines) together with the optimal execution strategy $u^{i,\star}$ (black dashed line). *Middle panels:* The five corresponding in-context example price impact trajectories $Y^{i,1} = \boldsymbol{I}_{\theta_i}(u^{i,1}), \ldots, Y^{i,5} = \boldsymbol{I}_{\theta_i}(u^{i,5})$. *Bottom panels:* The ICON surrogate operator prediction $\hat{\boldsymbol{I}}_{\theta_i}(u^{i,\star})$ (black dotted line) compared to the ground truth optimal price impact $Y^{i,\star} = \boldsymbol{I}_{\theta_i}(u^{i,\star})$ (red solid line).

$Y^\star = \boldsymbol{I}_\theta(u^\star)$ to obtain the exact output label, and compare it to the prediction of the ICON surrogate operator $\hat{\boldsymbol{I}}_\theta(u^\star)$ acting on $u^\star$. The relative ℓ^2 error under this setup is computed and visualized as heatmaps with varying hyperparameters in Figure 6, similar to the ones in Figure 5. This time, the error increases slightly, but still remains fairly small across the different ICON models and test datasets providing the context. We also observe that here, different compared to Figure 5, the errors depend on the push factor λ, simply because variations in λ alter the question condition $u^\star$.

To further elaborate on the previous analysis, we illustrate in Figure 7 five pairs of in-context examples $((u^{i,1}, Y^{i,1}), \dots, (u^{i,5}, Y^{i,5}))$ for $i = 1, 2, 3$ different propagator models with randomly sampled hyperparameters θ_i, generated as described in Section 4.2; the question condition $u^{i,\star}$ (i.e., the optimal execution strategy for model θ_i); and the prediction curves $\hat{\boldsymbol{I}}_{\theta_i}(u^{i,\star})$ compared to the ground truth price impact $\boldsymbol{I}_{\theta_i}(u^{i,\star})$. More precisely, each subfigure in Figure 7 corresponds to a specific model θ_i (exponential kernel in (a), non-singular power law kernel in (b), singular power law kernel in (c)). The top panel of each subfigure displays the in-context example conditions $u^{i,1}, \dots, u^{i,5}$ in colored solid lines, as well as the question condition $u^{i,\star}$ in black dotted lines. The middle panel shows the corresponding in-context example quantities of interest $Y^{i,1} = \boldsymbol{I}_{\theta_i}(u^{i,1}), \dots, Y^{i,5} = \boldsymbol{I}_{\theta_i}(u^{i,5})$. The bottom panel compares the in-context prediction $\hat{\boldsymbol{I}}_{\theta_i}(u^{i,\star})$ (red solid line) of the ICON model (trained on the dataset `ode` in (a), `ker` in (b), `sker` in (c), respectively) with the ground truth $\boldsymbol{I}_{\theta_i}(u^{i,\star})$ (black dotted line). Note that the optimal execution strategies $u^{i,\star}$ are almost flat for the three different propagator kernels with (randomly picked) hyperparameters θ_i. Remarkably, we observe that the three ICON surrogate operators are indeed capable of accurately predicting the corresponding price impact incurred by the selling rates $u^{1,\star}, u^{2,\star}, u^{3,\star}$, which gradually build up at different speeds and magnitudes, depending on the different propagator kernels and their hyperparameters $\theta_1 = (\beta_1, \lambda_1)$ (for the exponential kernel), $\theta_2 = (\gamma_1, \lambda_2)$ (for the non-singular power law kernel) and $\theta_3 = (\gamma_2, \lambda_3)$ (for the singular power law kernel), all of which are correctly inferred by the ICON models from the five prompted in-context examples.

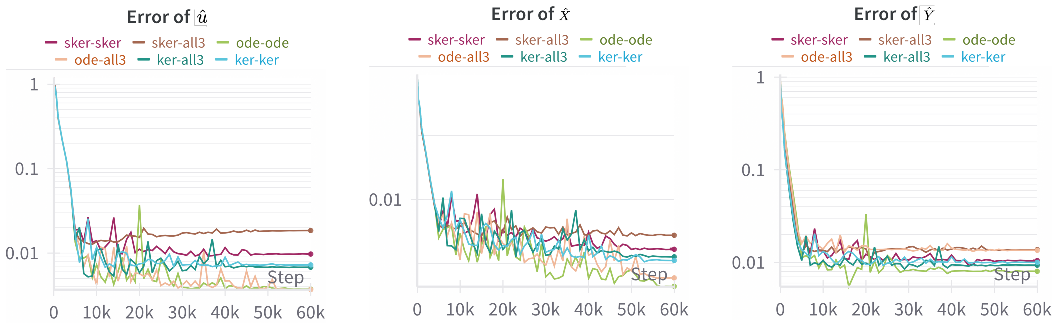
4.4 ICON-OCnet performance

We now illustrate the performance of our ICON-OCnet method described in Section 3.2. Specifically, using a pretrained ICON model as described above (trained on `ode`, `ker`, `sker`, `all3`, respectively, and prompted with 5 in-context examples from a specific propagator model θ) as a surrogate operator $\hat{\boldsymbol{I}}_\theta$ in the optimal execution problem, we perform a policy gradient method directly on the objective function in (13), and train a simple feed-forward neural network[4] with 60,000 iterations for the optimal order execution task. The initial inventory x expressed in ADV is sampled from a uniform distribution $U([0.01, 0.2])$. We compare the obtained OCnet policy $\hat{u}$ (with corresponding inventory $\hat{X}$ and price impact process $\hat{Y}$) to the actual ground truth optimal execution strategy $u^\star$ (with state processes $X^\star, Y^\star$) of the propagator model θ generating the in-context examples, which is computed via the implementation provided by [AJN25].

The relative ℓ^2 errors of selling rate, inventory, and price impact process (compared with the ground truth) versus the number of training steps are illustrated in Figure 8. We observe convergence of the policy gradient method after 60,000 iterations. In fact, the errors already drop to an acceptable level after 20,000 steps.

Table 2 summarizes the relative error of the objective function value in (13) of the trained OCnet policy $\hat{u}, \hat{Y}, \hat{X}$ with respect to the actual value of the objective function of the ground truth optimal policy $u^\star, Y^\star, X^\star$. The three different ICON models (columns) are trained on a

[4]The OCnet neural network structure consists of 2 hidden layers, 128 nodes in each layer, and GELU activation function.



(a) selling rate $\hat{u}$ (b) inventory $\hat{X}$ (c) price impact $\hat{Y}$

Figure 8: Relative errors of $\hat{u}, \hat{X}, \hat{Y}$ versus training iterations for different ICON models (second label) used as surrogate operator in the optimal execution problem with a specific propagator model θ providing the in-context examples (first label). The error values are mean values over 16 randomly selected hyperparameters θ and initial inventories x.

single correct dataset (i.e., the prompted in-context examples in the optimization problem are originating from the same propagator kernel type on which the ICON model was pre-trained). We observe that the error values are impressively small. This confirms the effectiveness of our proposed ICON-OCnet approach in finding the optimal execution strategy via few-shot in-context learning with ICON. Sample trajectories of the obtained OCnet selling rate $\hat{u}$ and corresponding price impact $\hat{Y}$ compared to the ground truth optimal execution strategy $u^\star$ and impact process $Y^\star$ for different propagator kernels (with randomly chosen hyperparameters θ and initial inventory x) are shown in Figure 9.

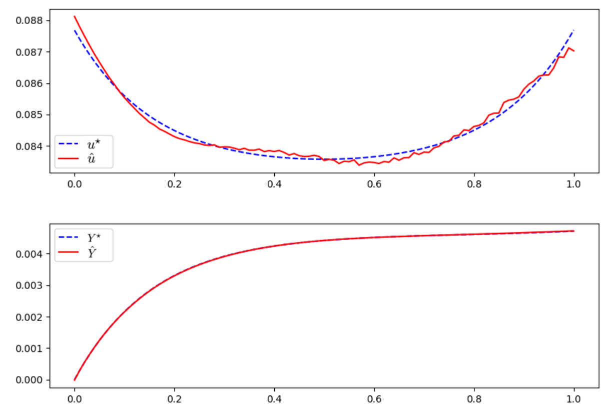
	`ode`	`ker`	`sker`
ICON-OCnet	5.80×10^{-8}	6.91×10^{-7}	4.55×10^{-7}

Table 2: Relative error of the optimal control objective function value for ICON models trained on a single correct dataset (rows). The values are averaged over 16 sets of randomly selected hyperparamters θ and initial inventories x.

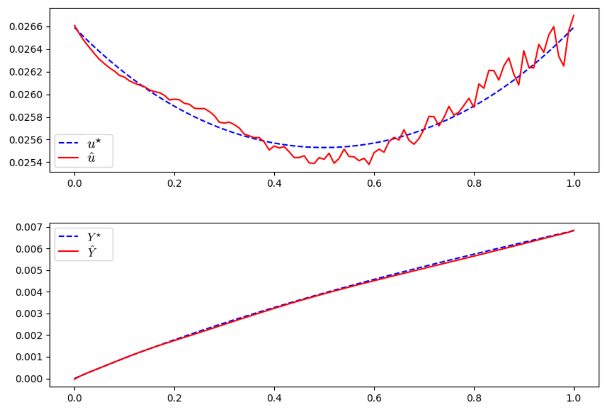
Lastly, we illustrate the influence of the hyperparameters on the error of ICON-OCnet in the heatmaps in Figures 10 and 11. Similarly to the heatmaps in Section 4.3, we split the range of each hyperparameter into 6 intervals, and then randomly sample in each box 16 sets of hyperparameters (and initial inventories x), for which we train an OCnet policy with the pretrained ICON model (trained on a single dataset in Figure 10 or on the mixed dataset in Figure 11), prompted with in-distribution examples as context, and compute the relative ℓ^2 error of $\hat{u}, \hat{X}, \hat{Y}$ with respect to the corresponding ground truths $u^\star, X^\star, Y^\star$. Overall, the errors are fairly small. As expected, the errors increase slightly when ICON is trained on the mixed dataset `all3`.

5 Conclusion and outlook

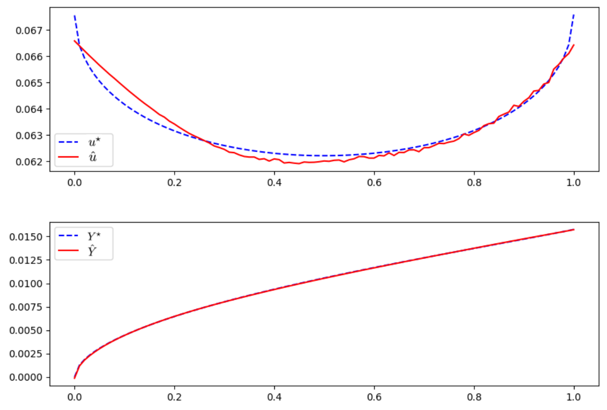
We introduced ICON-OCnet, a new approach to tackle an optimal stochastic control problem with an unknown operator mapping the control to the controlled state dynamics, which is inferred from examples. To this end, we used In-Context Operator Networks (ICON), a transformer-based neural network architecture introduced in [YLMO23], enhancing data ef-



(a) exponential kernel

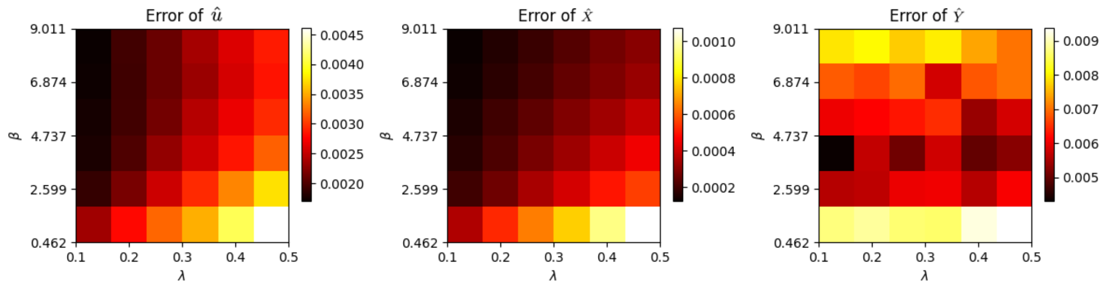


(b) non-singular power law kernel

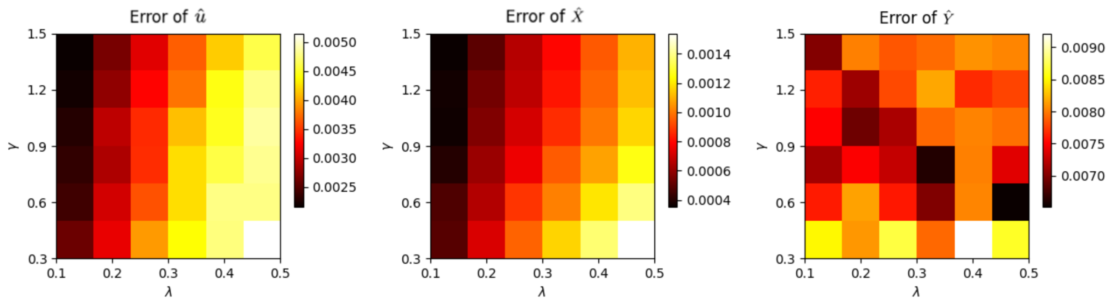


(c) singular power law kernel

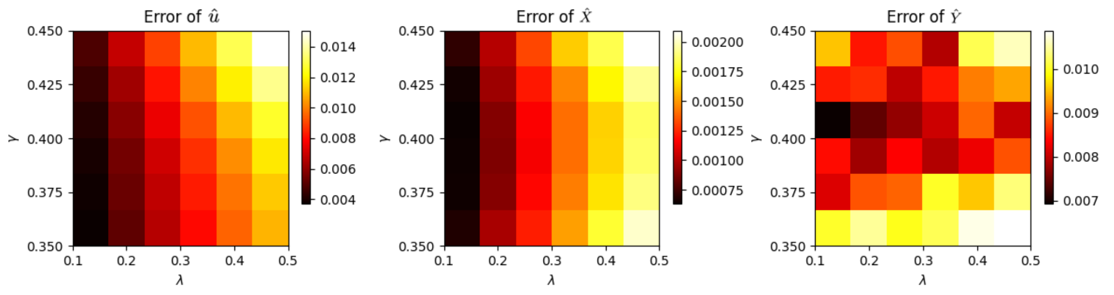
Figure 9: Example trajectories of the OCnet policy $\hat{u}$ with corresponding $\hat{Y}$ (red) compared to the ground truth $u^\star, Y^\star$ (blue) for different propagator kernels with randomly chosen hyperparameters θ and initial inventory x.



(a) exponential kernel

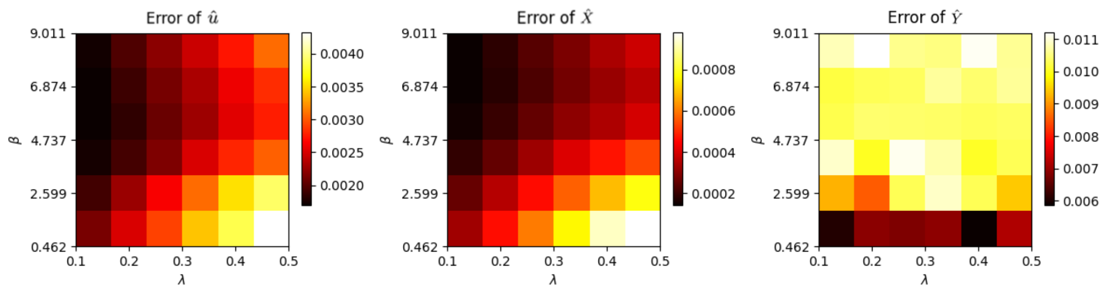


(b) non-singular power law kernel

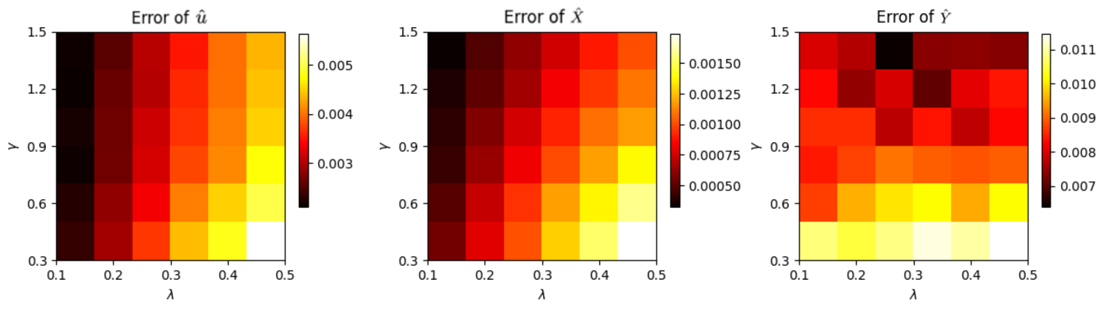


(c) singular power law kernel

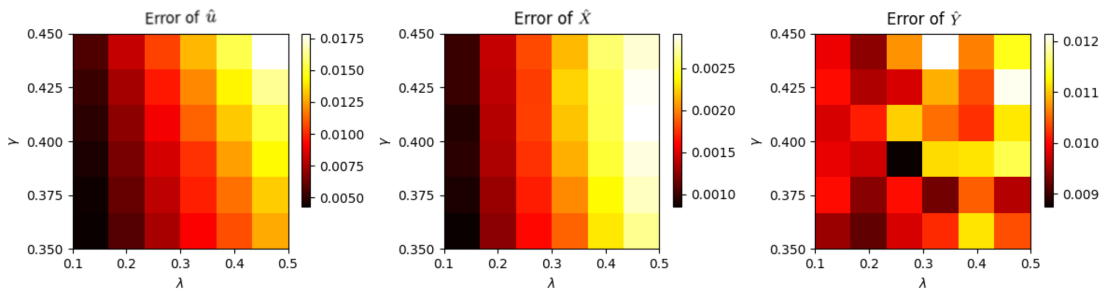
Figure 10: Heatmap for the relative error of the ICON-OCnet policy $\hat{u}$ and corresponding controlled state processes $\hat{X}, \hat{Y}$ with ICON trained on a single dataset (`ode` top, `ker` middle, `sker` bottom). The prompted in-context examples for initializing ICON are in-distribution from `ode_t` (top), `ker_t` (middle), `sker_t` (bottom).



(a) exponential kernel



(b) non-singular power law kernel



(c) singular power law kernel

Figure 11: Heatmap for the relative error of the ICON-OCnet policy $\hat{u}$ and corresponding controlled state processes $\hat{X}, \hat{Y}$ with ICON trained on the mixed dataset `all3`. The prompted in-context examples for initializing ICON are in-distribution from `ode_t` (top), `ker_t` (middle), `sker_t` (bottom).

ficiency and model flexibility due to their few-shot and transfer learning capabilities. This paper provides a proof of concept for this general methodology and showcases its efficiency in finding optimal order execution strategies via in-context learning. Specifically, we use trading rates and incurred price impact trajectories as examples to learn the price impact operator (step 1), and then learn the optimal liquidation strategy based on the learned operator (step 2). The price impact environment is unknown and inferred from data prompts (context). We effectively benchmarked our approach against ground-truth solutions for linear propagator models from [AJN25]. ICON-OCnet offers inference with reduced data requirements by merging offline pre-training with online few-shot learning, finding the model that is closest to the prompted examples and eliminating the need for retraining when new contexts arise.

Our work paves the way for future research in several directions: First, it is very sensible to test the performance of ICON-OCnet on a larger class of price impact models, including nonlinear models studied in [AJBDC+25]. This requires a more involved benchmarking procedure since ground-truth optimal execution strategies are no longer explicitly available but can only be computed approximately via numerical iterative schemes. Moreover, it is very interesting to investigate the training of ICON based on noisy in-context time series sample data pairs (u, P) as described in Section 3.1.2, including non-martingale dynamics for the unaffected price process, rather than on exact trajectory pairs (u, Y). This will require a much larger training set and more in-context examples to filter out the price impact. It might also require some degree of regularization to prevent overfitting. In a similar vein, we performed our offline training and online few-shot prompting using synthetic data. Of course, it would be interesting to study the training of ICON on real trade execution data. More precisely, due to the potential scarcity of the latter, it is conceivable to study the offline training on synthetic model-based data augmented by historical trade execution data, as well as using real trade execution data for the online in-context few-shot inference. However, this requires proprietary trading data on meta-order execution, which is notoriously hard to come by. Last but not least, it is also very appealing to study our sample-based in-context learning methodology for general path-dependent stochastic optimal control problems with general noise-driven state dynamics.

Acknowledgement

We would like to express our gratitude to Liu Yang for valuable discussions. T.M. and S.O. are supported by ONR MURI N00014-20-1-2787.

References

[AAB+19] Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural Operator: Graph Kernel Network for Partial Differential Equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2019.

[AJBDC+25] Eduardo Abi Jaber, Alessandro Bondi, Nathan De Carvalho, Eyal Neuman, and Sturmius Tuschmann. Fredholm Approach to Nonlinear Propagator Models, 2025.

[AJN25] Eduardo Abi Jaber and Eyal Neuman. Optimal liquidation with signals: The general propagator case. *Mathematical Finance*, 35(4):841–866, 2025.

[Ala18] Jay Alammar. The illustrated transformer. https://jalammar.github.io/illustrated-transformer/, 2018. Blog post; written June 27, 2018 (accessed Nov 9th, 2025).

[BBDG18] Jean-Philippe Bouchaud, Julius Bonart, Jonathan Donier, and Martin Gould. *Trades, quotes and prices: financial markets under the microscope*. Cambridge University Press, 2018.

[BDG24a] Fred Espen Benth, Nils Detering, and Luca Galimberti. Pricing options on flow forwards by neural networks in a Hilbert space. *Finance and Stochastics*, 28(1):81–121, 2024.

[BDG24b] Fred Espen Benth, Nils Detering, and Luca Galimberti. Structure-informed operator learning for parabolic Partial Differential Equations, 2024. arXiv:2411.09511.

[BFL09] Jean-Philippe Bouchaud, J. Doyne Farmer, and Fabrizio Lillo. How Markets Slowly Digest Changes in Supply and Demand. In Thorsten Hens and Klaus Reiner Schenk-Hoppe, editors, *Handbook of Financial Markets: Dynamics and Evolution*, Handbooks in Finance, pages 57–160. North-Holland, San Diego, 2009.

[BGPW04] Jean-Philippe Bouchaud, Yuval Gefen, Marc Potters, and Matthieu Wyart. Fluctuations and response in financial markets: the subtle nature of 'random' price changes. *Quantitative Finance*, 4(2):176–190, 2004.

[CJP15] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. *Algorithmic and High-Frequency Trading*. Cambridge University Press, 1 edition, October 2015.

[Gat10] Jim Gatheral. No-dynamic-arbitrage and market impact. *Quantitative Finance*, 10(7):749–759, 2010.

[Gué16] Olivier Guéant. *The Financial Mathematics of Market Liquidity*. New York: Chapman and Hall/CRC, 2016.

[HE16] Jiequn Han and Weinan E. Deep learning approximation for stochastic control problems, 2016. arXiv:1611.07422.

[HMMKW25] Natascha Hey, Iacopo Mastromatteo, Johannes Muhle-Karbe, and Kevin Webster. Trading with concave price impact and impact decay—theory and evidence. *Oper. Res.*, 73(3):1230–1247, May 2025.

[KKSH24] Soohan Kim, Jimyeong Kim, Hong Kee Sul, and Youngjoon Hong. An adaptive dual-level reinforcement learning approach for optimal trade execution. *Expert Systems with Applications*, 252:124263, 2024.

[KLL$^+$22] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. *Journal of Machine Learning Research*, pages 1–97, 2022.

[LJP$^+$21] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[LN19] Charles-Albert Lehalle and Eyal Neuman. Incorporating signals into optimal trading. *Finance and Stochastics*, 23(2):275–311, 2019.

[NSZ23] Eyal Neuman, Wolfgang Stockinger, and Yufei Zhang. An Offline Learning Approach to Propagator Models, 2023. arXiv:2309.02994.

[NV22] Eyal Neuman and Moritz Voß. Optimal signal-adaptive trading with temporary and transient price impact. *SIAM Journal on Financial Mathematics*, 13(2):551–575, 2022.

[NWZ23] Marcel Nutz, Kevin Webster, and Long Zhao. Unwinding Stochastic Order Flow: When to Warehouse Trades, 2023. arXiv:2310.14144.

[NZ23] Eyal Neuman and Yufei Zhang. Statistical Learning with Sublinear Regret of Propagator Models, 2023. arXiv:2301.05157.

[OW13] Anna A. Obizhaeva and Jiang Wang. Optimal trading strategy and supply/demand dynamics. *Journal of Financial Markets*, 16(1):1 – 32, 2013.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[Web23] Kevin T. Webster. *Handbook of Price Impact Modeling.* Chapman and Hall/CRC, 2023.

[YLMO23] Liu Yang, Siting Liu, Tingwei Meng, and Stanley J. Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.

[YLO25] Liu Yang, Siting Liu, and Stanley J. Osher. Fine-tune language models as multi-modal differential equation solvers. *Neural Networks*, 188:107455, 2025.