A More Implementation Benchmarks

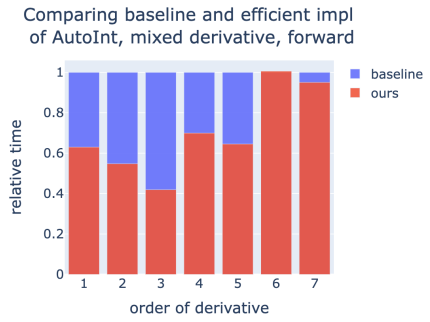

Figure 5: Forward mixed $(d/dx_1dx_2...dx_k)$ partial derivative computation speed, 3 layers MLP

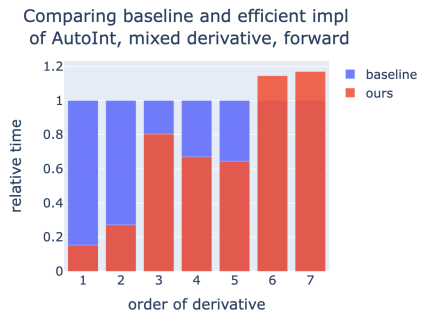

Figure 6: Forward mixed $(d/dx_1dx_2...dx_k)$ partial derivative computation speed, 4 layers MLP

As the number of MLP layers increases, the low order derivative computation becomes faster (relative to PyTorch naive implementation), where as the higher order derivative computation becomes slower. This is because our implementation uses the Python for loop. Our implmentation is faster than the baseline in most of the case.

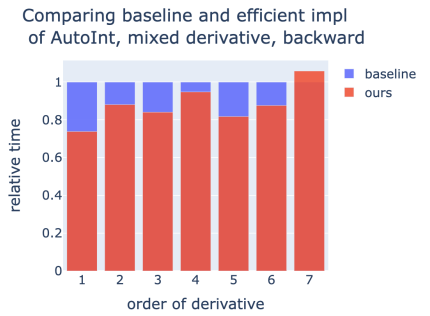

Figure 7: Forward + Backward mixed $(d/dx_1dx_2...dx_k)$ partial derivative computation speed, 3 layers MLP

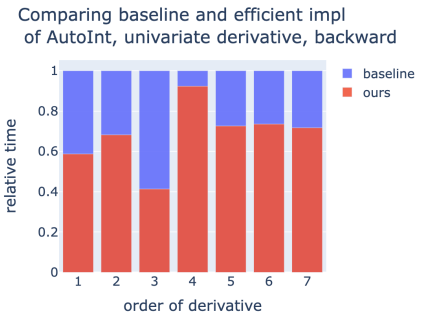

Figure 8: Forward + Backward univariate $(d/dx_1dx_1...dx_1)$ partial derivative computation speed, 3 layers MLP

When considering the backward time as well as the forward time, our implementation is still steadily faster than the baseline. It is worth noting that our method is even faster when calculating univariate partial derivative instead of mixed partial derivative, because the number of python for-loop iterations is small.

B STSC and ST-Hawkes Introduction and Simulation Parameters

We use the same parameters as Zhou et al. [2022].

The STSCP's and the STHP's kernels $g_0(\mathbf{s})$ and $g_2(\mathbf{s}, \mathbf{s}_j)$ are prespecified to be Gaussian:

$$g_0(\mathbf{s}) := \frac{1}{2\pi}|\Sigma_{g0}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{s}-[0,0])\Sigma_{g0}^{-1}(\mathbf{s}-[0,0])^T\right)$$
$$g_2(\mathbf{s}, \mathbf{s}_j) := \frac{1}{2\pi}|\Sigma_{g2}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{s}-\mathbf{s}_j)\Sigma_{g2}^{-1}(\mathbf{s}-\mathbf{s}_j)^T\right)$$

The STSCP is defined on $\mathcal{S} = [0,1] \times [0,1]$, while the STHP is defined on $\mathcal{S} = \mathbb{R}^2$. The STSCP's kernel functions are normalized according to their cumulative probability on $\mathcal{S}$. Table 3 shows the

simulation parameters. The STSCP's spatial domain is discretized as an 101×101 grid during the simulation.

Table 3: Parameter settings for the synthetic dataset

		α	β	μ	Σ_{g0}	Σ_{g2}
ST-Hawkes	DS1	.5	1	.2	[.2 0; 0 .2]	[0.5 0; 0 0.5]
	DS2	.5	.6	.15	[5 0; 0 5]	[.1 0; 0 .1]
	DS3	.3	2	1	[1 0; 0 1]	[.1 0; 0 .1]
ST-Self Correcting	DS1	.2	.2	1	[1 0; 0 1]	[0.85 0; 0 0.85]
	DS2	.3	.2	1	[.4 0; 0 .4]	[.3 0; 0 .3]
	DS3	.4	.2	1	[.25 0; 0 .25]	[.2 0; 0 .2]

C Model Setup Details

We list out the detailed hyperparameter settings in Table 4. The same set parameters are used across all datasets, except the learning rate. This demonstrate our model is robust to hyperparameters.

Name	Value	Description
Optimizer	Adam	-
Learning rate	-	Depends on dataset, [0.0002, 0.004]
Momentum	0.9	Adam momentum
Epoch	50 / 100	50 for synthetic dataset and 100 for real-world dataset
Batch size	128	-
Activation	tanh	Activation function in L and M (intensity parameter networks)
nprodnet	2 / 10	Number of product nets to sum in L and M 2 for synthetic dataset and 10 for real-world dataset
bias	true	L and M use bias in their linear layers

Table 4: Hyperparameter settings for training `AutoSTPP` on all datasets.

D Forward-pass Algorithm for Automatic Integration

Function: dnforward(f, n, x, dims), partition(n, k) finds all k-subset partitions of n

Data: n, dimension of $f(x)$, x, a tensor of shape (batch, dim),
$dims$, list of dimensions to derive, $layers$, composite functions in f
Result: $d^{dims}f/dx^{dims}$
Initialize dictionary dnf, mapping from $dims$ to $d^{dims}f/dx^{dims}$, empty list pd
if *$dims$ is one dimensional* **then**
  Compute $f(x)$, store intermediate outputs to dictionary
  pd ← [all zeros but one at deriving dimension]
else
  for *subdims ∈ combination(dims, len(dims)-1)* **do**
    Call dnforward(n, x, subdims), store intermediate outputs to dictionary
  end
  pd ← [all zeros]
end

```
for layer ∈ layers do
    if layer is linear then
        pd append last pd ×W^T, W is the linear weight
    else if layer is activation then
        if dims is one dimensional then
            termsum ← last pd × activation first order derivative with input f
        else
            termsum ← 0
            for order ∈ 0... len(dims) do
                if order = 0 then
                    term ← last pd
                else
                    term ← 0
                    for part ∈ partition(dims, order + 1) do
                        temp ← 1. for subdims ∈ part do
                            temp ← temp × dictionary value for key subdims
                            term ← term + temp
                        end
                        termsum ← termsum + term
                    end
                end
                termsum ← termsum × activation nth order derivative with input f
            end
        end
    pd append termsum
end
return last pd
```

E Universal Approximation Theorem for Derivative Network

Consider an AutoInt integral network with the form

$$g(x) = C \cdot (\sigma \circ (A \cdot x + b)), \qquad A \subseteq \mathbb{R}^{k \times n}, b \subseteq \mathbb{R}^k, C \subseteq \mathbb{R}^k$$

where σ denotes a $\mathbb{R} \to \mathbb{R}$ continuous non-polynomial function applies elementwise to each component of input.

The derivative network thus takes the form

$$g'(x) = C \cdot (\sigma' \circ (A \cdot x + b) \circ A_{col}),$$

where $A_{col} \subseteq \mathbb{R}^k$ is a column of A that corresponds to the deriving dimension.

Recall the universal approximation theorem [Daniels and Velikova, 2010], which says for every compact $K \subseteq \mathbb{R}^n$ and $f \in C(K, \mathbb{R}), \varepsilon > 0$, there exist A, b, C such that

$$\sup_{x \in K} \|f(x) - g(x)\| < \varepsilon$$

Proposition E.1. *(Universal Approximation Theorem for Derivative Network) for every compact $K \subseteq \mathbb{R}^n$ and $f \in C(K, \mathbb{R}), \varepsilon > 0$, there exists $A \in \mathbb{R}^{k \times n}, b \in \mathbb{R}^k, C \in \mathbb{R}^k, \beta \in \mathbb{R}$ such that*

$$g(x) := C \cdot (\sigma \circ (Ax + b)) - \beta x$$
$$\sup_{x \in K} \|f(x) - g'(x)\| < \varepsilon$$

Proof. Given the mapping f, by UAT there exists A, b, C that approximate $f(x)$. Construct $\tilde{C} \in \mathbb{R}^k$ and $\beta \in \mathbb{R}$, such that

$$\tilde{C}_j = \begin{cases} C_j / A_{col,j}, & A_{col,j} \neq 0 \\ 0, & A_{col,j} = 0 \end{cases}, \qquad \text{and} \quad \beta = \sum_{j | A_{col,j} = 0} C_j \sigma(b_j)$$

Then,

$$
\begin{aligned}
&\sup_{x\in K} \|f(x) - C\cdot(\sigma\circ(A\cdot x+b))\| \\
&= \sup_{x\in K} \left\| f(x) - \sum_{j=1}^{k} C_j(\sigma(A_j\cdot x+b_j)) \right\| \\
&= \sup_{x\in K} \left\| f(x) - \sum_{j=1}^{k} \tilde{C}_j(\sigma(A_j\cdot x+b_j)A_{col,j}) - \sum_{j|A_{col,j}=0} C_j\sigma(b_j) \right\| \\
&= \sup_{x\in K} \|f(x) - \tilde{C}\cdot(\sigma'\circ(A\cdot x+b)\circ A_{col}) - \beta\|,
\end{aligned}
$$

Note that $\tilde{C}\cdot(\sigma'\circ(A\cdot x+b)\circ A_{col})-\beta = \dfrac{d}{dx_{col}}\Big(\tilde{C}\cdot(\sigma\circ(Ax+b))-\beta x\Big)$, which is the derivative net of a two-layer feedforward integral network.

F Relationship between Number of ProdNets and Model Expressivity

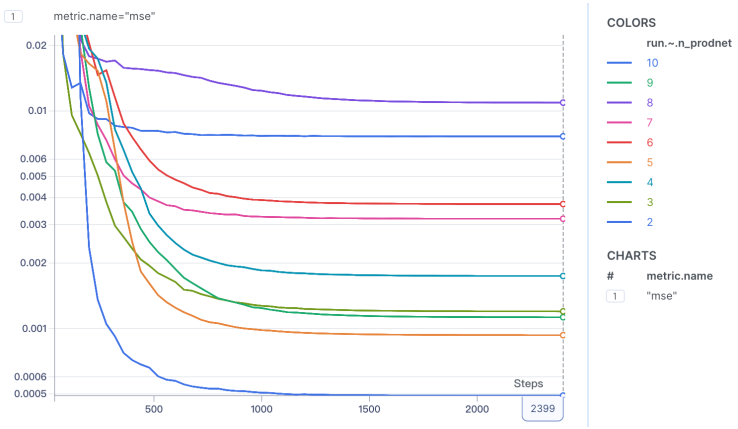

Figure 9: Training MSE for fitting a positive derivative network to $\sin(x)\cos(y)\sin(z)+1$

We applied ten summations of positive ProdNets to fit the non-multiplicative-decomposable function $\sin(x)\cos(y)\sin(z)+1$. Each ProdNet consisted of three MLP components, each with two hidden layers of 128 dimensions. All models were trained with a fixed learning rate of 0.005.

Our results, shown in Figure 9, indicate that increasing the number of ProdNets generally improves the model's performance in fitting the non-decomposable function. The model with the best MSE was produced by using 10 ProdNets, while the model with 2 ProdNets had the second-worst performance. This is intuitively sensible, as more linear terms are typically required to precisely express an arbitrary function. However, we observed that employing more ProdNets does not always lead to better performance, as demonstrated by the model with 8 ProdNets.