AuctioNN: A Simulation-Driven Bidding Engine for Large-Scale Cost-Per-Action Optimization

Param Kapur, Cameron Adams, Quinn Behrens, Ernie Zhang
{paramkapur, cadams, qbehrens, irenez}@reed.edu

May 5, 2025

Abstract

AuctioNN is a sandbox framework and decision engine that evaluates whether a *neural-network–guided bidding policy* can reduce system-wide cost-per-action (CPA) while meeting campaign-specific goals and operational constraints. The instrumented simulator streams impression requests one at a time, emulating an external, first-price and second-price ad exchange. We rigorously compare the proposed policy against heuristic baselines using *marginal CPA*, total conversions, and revenue uplift. Results generalize to media owners such as *iHeart* that simultaneously monetize their owned&operated (O&O) inventory and purchase incremental reach on open exchanges. The full architecture, mathematical formulation, and open research questions are documented here.

1 Introduction

Online advertising operates at a staggering scale, with billions of individual auctions happening every day across the web, apps, and streaming platforms. At the core of this system lies the real-time bidding (RTB) process, where advertisers compete to show their ads to specific users at specific moments. But not all impressions are created equal, some users are far more likely to engage, click, or convert than others. The challenge, then, is to bid intelligently: to spend money where it is most likely to drive meaningful outcomes, such as purchases, signups, or app installs, while respecting the constraints of campaign budgets and performance goals.

Large media publishers increasingly extend their reach by *buying* impressions on open ad exchanges. Operating 10^3–10^4 concurrent campaigns in real time raises three entangled challenges:

1. **Win the *right* inventory**—impressions that drive post-click or post-view conversions.
2. **Respect campaign goals**—especially CPA targets stipulated in insertion orders.
3. **Obey operational constraints**—budgets, pacing, and user-level frequency capping.

This work posits that a neural model, combined with explicit budget/frequency bookkeeping, can outperform classical heuristics that ignore conversion likelihood, or at the very least, random bidding.

Traditionally, bidding strategies in RTB have either relied on simple heuristics, such as bidding proportionally to an advertiser's declared value per conversion, or leveraged machine learning models that directly output a recommended bid price based on the enviroment. Much of the recent work in this space has focused on reinforcement learning (RL) approaches, where an agent learns a

bidding policy through interaction with an environment, balancing exploration with campaign-level constraints like budget or cost-per-action (CPA) targets.

Our approach takes a different path. Rather than asking the neural network to output a bid price directly, we use it to focus on *what we believe to be the root of the problem*: predicting the likelihood that a given impression will result in a conversion. In other words, we train a neural network purely as a *conversion predictor*, separate from the bidding logic itself. We then use this predicted probability as an input to a decision (auction) algorithm that determines whether to bid, on which campaign, and at what price.

The key motivation for this work is that accurate conversion estimation can enable more principled and interpretable bidding decisions. This separation of concerns allows the model to specialize in what it does best (learning conversion patterns from data) while keeping the bidding policy flexible and transparent.

In this paper, we present *AuctioNN*, a simulation driven bidding engine that integrates neural network based conversion prediction into the bidding process. We compare our system against heuristic baselines and explore whether this two-stage approach can outperform strategies that directly optimize the bid price. Along the way, we hope to clarify how learning better *conversion likelihoods*, rather than directly learning *bids*, can lead to simpler, scalable, and more interpretable auction strategies.

2 Background

2.1 The Landscape of Bidding in Online Advertising

At the heart of real-time bidding (RTB) systems is a deceptively simple question: *how much should an advertiser bid for a given ad impression?* Yet behind this question lie several layers of complexity. The auction mechanisms themselves may vary (first-price, second-price, programmatic guaranteed), campaign objectives often include a mix of budget limits and cost-per-action (CPA) goals, and, critically, the true value of an impression (whether it will lead to a conversion) is rarely known at the time of bidding.

Historically, the first generation of bidding strategies relied on *rule-based heuristics*. These approaches often computed the expected value of an impression by estimating the probability of a click (CTR) or conversion (CVR) and then multiplying this probability by the advertiser's declared value per click or per conversion. The bid was then set either directly to this expected value or scaled by a constant factor to account for risk preferences or auction mechanics. These heuristics worked well in the early days of second-price auctions, where truthful bidding (bidding your true value) is an optimal strategy under ideal conditions.

However, with the majority of online auctions being *first-price auctions*, where the winning bidder pays their own bid amount, disrupted this simplicity. In first-price environments, bidding your full value results in systematically overpaying, so bidders must "shade" their bids downward to avoid eroding profit margins. This need for bid shading introduced a new layer of strategic uncertainty, where bidders had to model not just their own value but also anticipate the distribution of competing bids.

2.2 How an Open Ad Exchange Works

For each page-view or app launch, the exchange issues a *bid request* containing contextual metadata (timestamp, device, geo, etc.). Every buyer has roughly 100 ms to (i) decide whether to participate and (ii) transmit a bid price. The highest bid wins and immediately pays the *clearing price*.

In most modern exchanges this is a *first-price auction*; the winner pays its own bid. However, some exchanges use different auction models like *second-price* or programmatic guaranteed (PG) auctions. Any downstream *conversion event* (purchase, signup, ...) is reported asynchronously, often minutes to days later.

2.3 Reinforcement Learning for Bidding: Direct Bid Optimization

As these complexities grew, researchers began turning to *reinforcement learning (RL)* to optimize bidding strategies. In this framework, bidding is modeled as a sequential decision-making problem, where the agent interacts repeatedly with an environment (the auction marketplace), and learns a policy that maximizes long-term rewards (e.g., total conversions, clicks, or revenue) while respecting budgetary or pacing constraints.

One early influential approach was the *Real-Time Bidding by Reinforcement Learning (RLB)* framework by Cai et al. (2017). This work framed bidding as a Markov Decision Process (MDP), where the state included remaining budget, remaining time, and other campaign features. The action space was the set of possible bid prices. The RLB agent used value-based RL methods to learn the expected cumulative reward from each state-action pair and selected bids accordingly. The reward function was typically tied to immediate outcomes like clicks or conversions.

Following this, Wu et al. (2018) addressed an important limitation of the basic RLB approach: naïve reward structures can lead to poor budget pacing, with agents spending too aggressively early in a campaign or failing to adapt to delayed conversion feedback. To remedy this, Wu et al. introduced *budget-aware state representations* and carefully designed reward functions that penalized over- or under-spending relative to pacing goals. Their approach allowed the RL agent to account for the long-term impact of current bidding decisions on campaign success, making it more robust in practice.

Further advancements leveraged *actor-critic methods* and *policy gradient techniques*, which provided greater flexibility in handling continuous bid values and more stable learning dynamics. For example, Liu et al. (2021) introduced a *Soft Actor-Critic (SAC)* approach to bidding, which incorporated entropy regularization to encourage exploration and avoid premature convergence to suboptimal bidding strategies. Instead of outputting a hard bid price directly, their system learned a distribution over bid adjustment factors, allowing for more nuanced bidding behavior under uncertainty.

In many of these RL-based systems, the *neural network learns the bidding policy itself*, meaning that the network outputs the bid amount or bid adjustment as a function of the input state. Conversion probabilities may still be implicitly learned within the network, but they are not exposed directly as outputs or decision-making inputs. Instead, the network collapses prediction and decision-making into a single policy.

2.4 Conversion Prediction as a Separate Task

A parallel thread of research in online advertising focuses on *conversion rate prediction (CVR)* and *click-through rate prediction (CTR)* as standalone supervised learning tasks. These models, often logistic regression or gradient-boosted trees, take impression features (user metadata, time of day, device type, etc.) and predict the likelihood that a given impression will lead to a conversion.

The outputs of these models are typically used as *inputs into heuristic bidding rules*. A common strategy is to bid proportionally to the predicted conversion probability times the advertiser's declared value per conversion. For example, if the CVR is 1% and the conversion is worth \$100, the bid might be \$1 (or shaded downward in a first-price setting).

Google's *Smart Bidding* strategies, as well as Facebook's auction system, are examples of large-scale production systems where machine learning predicts the likelihood of various outcomes and then uses those predictions to guide bidding decisions. However, in many of these systems, the predictive model itself is *deeply coupled* with the bidding logic, and the specifics of how conversion probabilities are used within the bid calculation are proprietary. This is the path our model will utalize.

2.5 The Gap: Prediction vs. Policy

The central observation motivating our work is that *most reinforcement learning and direct policy-learning approaches collapse prediction and bidding into a single learned function*, whereas supervised conversion prediction models are typically used only within simple heuristic bidding formulas.

We believe there is value in keeping these two steps distinct.

Our work builds on the idea that *accurate prediction of conversion likelihood is a critical first step*, and that bidding should then be handled by a separate, explicitly defined decision engine. This separation has several advantages:

- **Interpretability**: The predicted conversion probability is a meaningful, transparent quantity that can be audited, analyzed, and improved independently of the bidding logic.
- **Flexibility**: The bidding strategy can be easily adjusted (e.g., to change the aggressiveness of bid shading or apply new campaign constraints) without retraining the prediction model.
- **Modularity**: Improvements in the conversion predictor (e.g., better features, better model architecture) directly enhance the quality of the inputs to the bidding engine, without requiring changes to the decision logic.

2.6 How Our Approach Fits In

In this paper, we propose *AuctioNN*, a bidding engine that explicitly separates *conversion prediction* from *bid decision-making*. AuctioNN uses a neural network to estimate the probability of conversion for each impression-campaign pair. This probability feeds into a bid selection rule that incorporates campaign-specific parameters such as target CPA, budget pacing, and frequency capping.

This approach is influenced by several lines of past research:

- From **heuristic bidding models**, we inherit the principle that bids should reflect expected value (predicted conversion probability times value per conversion).
- From **supervised learning approaches to CVR modeling**, we adopt the use of predictive models trained on historical data to estimate conversion likelihoods.
- From **reinforcement learning and auction theory**, we recognize the importance of long-term planning, pacing, and strategy under first-price auction dynamics.

However, we intentionally *do not use the neural network to directly output bids*, nor do we collapse the policy into the model itself. Instead, we leverage the predictive model as an informed signal within a broader bidding framework. This distinction allows us to maintain interpretability, modularity, and flexibility while still benefiting from the power of neural networks for learning complex patterns in the data.

3 System Architecture & Environment

AuctioNN treats the exchange as an online stream, delivering impression I_t at discrete time t. For every impression the engine must *(a)* decide whether to bid, *(b)* pick a campaign, and *(c)* set the bid price. Table 1 formalizes the symbols used throughout the paper.

Table 1: Core symbols and state variables.

Symbol / Term	Meaning
I_t	Impression delivered at discrete time t.
features(I_t)	All metadata included in the bid request.
C	Active campaign set. $c \in C$ is one specific campaign.
budget_remaining[c]	Dollars left for campaign c.
$p_{\text{conv}}(c, I_t)$	Predicted conversion probability if c wins I_t.
value_per_conv[c]	Advertiser-declared value of one conversion.
target_CPA[c]	Optional maximum CPA for campaign c.
ad_stock[user, c] or τ	Recent exposure score (models ad fatigue).
score(c, I_t)	Scalar used to select the winning campaign.
bid	Dollar price sent to the exchange.
β	Bid shading factor (initially 1.0).
clearing_price	Amount paid if the bid wins (= bid in first-price).
marginal CPA	ΔSpend/ΔConversions over a window.

3.1 End-to-End Decision Loop

The procedure below executes for *every* incoming impression.

Algorithm 1 AuctioNN per-impression decision loop

Require: incoming impression I_t

Receive I_t; extract features(I_t)
Eligibility gate:
  $C_{\text{elig}} \leftarrow \{c \in C \mid \texttt{budget\_remaining}[c] > 0 \ \wedge \ \texttt{ad\_stock}[\text{user}, c] < \tau\}$
for all $c \in C_{\text{elig}}$ **do**
  $p_{\text{conv}}(c, I_t) \leftarrow f_\theta\big(\text{features}(I_t), \text{embed}(c), \text{ad\_stock}[\text{user}, c]\big)$ ▷ see §4
  $$\text{score}(c, I_t) \leftarrow \begin{cases} \dfrac{p_{\text{conv}}}{\text{target\_CPA}[c]}, & \text{if target\_CPA exists,} \\ p_{\text{conv}} \times \text{value\_per\_conv}[c], & \text{otherwise.} \end{cases}$$
end for
$c^* \leftarrow \arg\max_c \text{score}(c, I_t)$
expected_value $\leftarrow p_{\text{conv}}(c^*, I_t) \times \text{value\_per\_conv}[c^*]$
bid $\leftarrow \beta \times$ expected_value ▷ $\beta = 1$ initially[1]
Transmit bid; observe win/loss and clearing_price
if win **then**
  budget_remaining[c^*] −= clearing_price
  ad_stock[user, c^*] += 1
end if
Log *(*I_t, c^*, bid, win/loss, time, utility*)* for offline analysis

3.2 Decision Loop Pipeline

The figure below illustrates the decision loop in a block diagram.

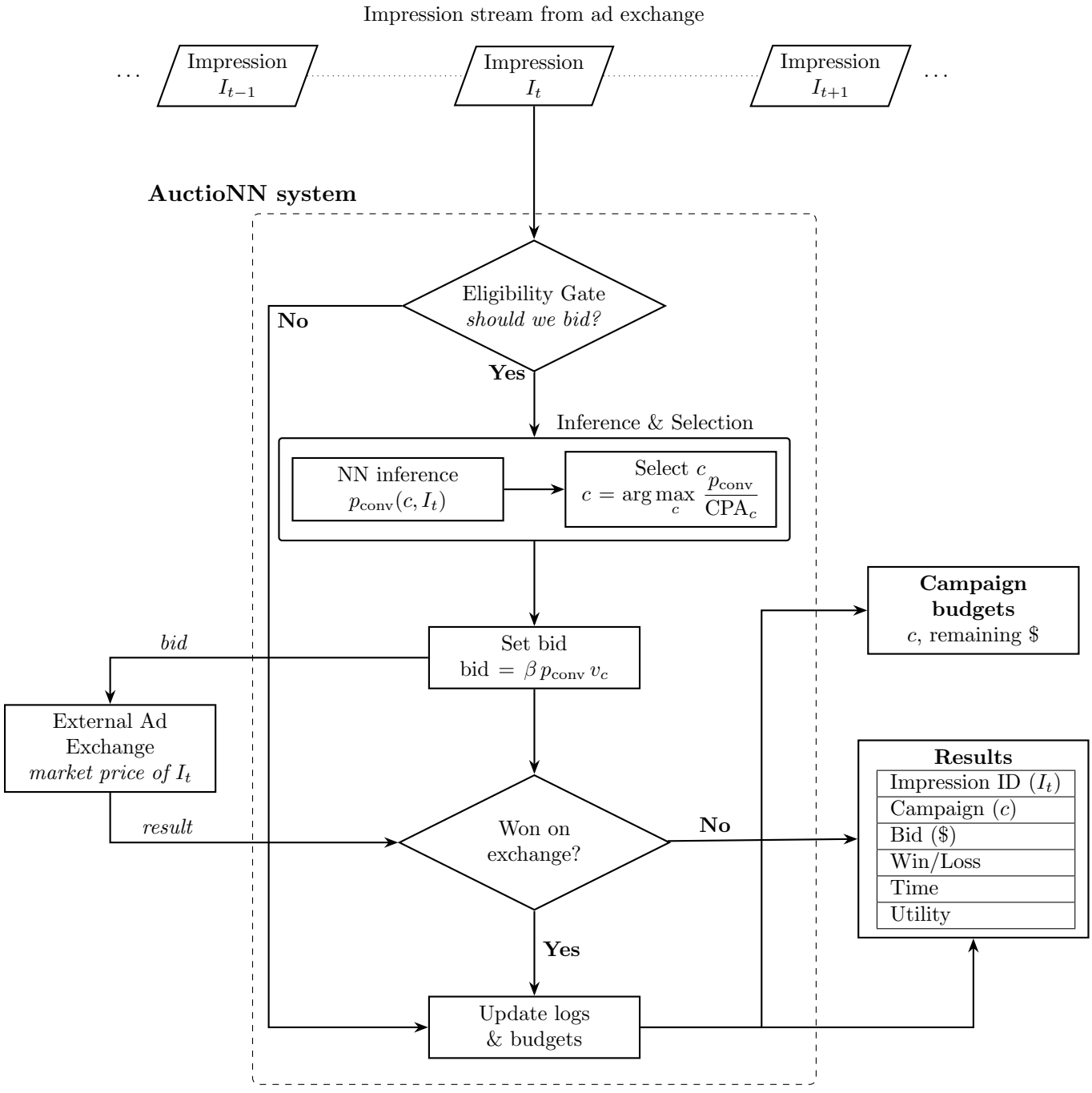

Figure 1: Fully connected block diagram of AuctioNN's *per-impression* decision loop, emphasising the continuous stream of impressions $I_{t-1}, I_t, I_{t+1}, \ldots$. The compound *Inference & Selection* stage first scores every campaign with a neural network and then chooses c via the displayed arg-max rule.

4 Neural Model

4.1 Feature Representation

We structure the impression data into two categories of features:

- **Categorical features (10 total):** Each categorical feature is first integer-encoded and then mapped into a 32-dimensional embedding vector. These include ad-placement details (e.g., placement ID, DMA, country), user-agent characteristics (e.g., browser, OS, device family), and crucially, the campaign identifier itself.
- **Numerical features (8 total):** We incorporate boolean flags indicating device attributes (e.g., mobile or tablet usage) and encode temporal signals—such as hour-of-day and day-of-week—using cyclical sine/cosine transformations. Each numerical feature is standardized via z-score normalization.

4.2 Network Architecture

The neural network itself is a compact Multi-Layer Perceptron (MLP) designed to predict the probability of conversion conditioned jointly on impression features and campaign identity. Specifically, the campaign ID is treated as just another categorical input, allowing the model to naturally learn campaign-specific conversion tendencies. During inference, we can easily swap in any campaign ID to predict how that campaign would perform for a given impression.

The network processes input as follows:

$$
\begin{aligned}
\mathbf{e} &= \bigoplus_{f \in \mathcal{C}} \mathrm{Embed}_f(x_f) && \in \mathbb{R}^{10\times 32} \\
\mathbf{z} &= [\, \mathbf{e} \parallel \mathbf{x}_{\mathrm{num}} \,] && \in \mathbb{R}^{328} \\
\mathbf{h}_1 &= \mathrm{ReLU}\big(\mathrm{BN}_1(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1)\big) \\
\mathbf{h}_2 &= \mathrm{ReLU}\big(\mathrm{BN}_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)\big) \\
\ell &= \mathbf{w}_{\mathrm{out}}^{\top}\mathbf{h}_2 + b_{\mathrm{out}} \\
p_{\mathrm{conv}} &= \sigma(\ell)
\end{aligned}
$$

The model comprises two hidden layers with 128 and 64 neurons, each followed by batch normalization and a dropout rate of 0.3, totaling roughly 55,000 parameters. It is trained using the Adam optimizer with a learning rate of 10^{-3} and a batch size of 2048 over 10 epochs. To handle the severe class imbalance in the data, we employ weighted binary cross-entropy loss (pos_weight = 137.0).

4.3 Current Performance

Table 2 summarizes the validation results obtained after training on a dataset of approximately 3.85 million impressions (15% validation split). **Please note that this dataset comprised of only ONE campaign. We are yet to train on all 100 campaigns.**

Table 2: Validation metrics after 10 epochs of training

Metric	Value
ROC–AUC	0.9295
PR–AUC (average precision)	0.4347
Log loss	0.2849
Accuracy @0.5	0.8896
Precision @0.5	0.0493
Recall @0.5	0.7791
F_1 @0.5	0.0928
True negatives (TN)	509 935
False positives (FP)	62 756
False negatives (FN)	923
True positives (TP)	3256

The model achieves excellent predictive quality (ROC–AUC >0.92), although precision remains challenging due to the high class imbalance inherent in conversion data.

4.4 Future Improvements

To further enhance the model's capability, we plan several improvements:

- Introduce a balanced data sampling strategy during training to ensure the model learns effectively across campaigns of varying sizes.
- Optimize inference by vectorizing model predictions, allowing scoring of all 100 candidate campaigns simultaneously instead of individually.
- Enhance training stability through adaptive learning rate schedules (ReduceLROnPlateau), early stopping criteria, and experimenting with focal loss to better handle extreme class imbalances.
- Regularize campaign embeddings (via L2 penalties) and introduce techniques such as negative sampling or data augmentation to improve generalization on sparse or unseen campaign-impression combinations.

5 Evaluation Methodology

We benchmark three policies: [WORK IN PROGRESS]

1. **Random**—bid a constant \$1 CPM on a random eligible campaign.
2. **Heuristic**—industry default: bid proportional to campaign value, ignoring p_{conv}.
3. **AuctioNN (proposed)**—full decision loop of §3.2.

5.1 Success Metrics

- **Marginal CPA** (ΔSpend/ΔConversions) relative to baseline.
- **Total conversions** aggregated across all campaigns.
- **Revenue uplift** (= advertiser value – media cost).
- **Pacing error**: |actual spend − planned spend|/planned spend ≤ 0.05.
- **Latency**: distributive $p_{95} \leq 2\,\text{ms}$ per impression on a MacBook M-series laptop.

6 Implementation Notes

6.1 Repository Layout

```
auctionn_sim/
  - campaign.py            % Campaign dataclass + bootstrapper
  - exchange.py            % ImpressionGenerator, OnlinePreprocessor, Market
  - decision_loop.py     % Algorithm 1 implementation
  - logger.py            % Parquet-backed ResultsLogger
  - models/                % best_model.pth  (TorchScript or state-dict)
  - preprocessors/         % encoder / scaler artefacts
  - main.py                % CLI entry point
```

6.2 Data Preprocessing

The data preprocessing is done in the preprocess.py file. The data is preprocessed in the following steps:

- Integer encoding of categorical features.
- Standardization of numerical features.

The data before preprocessing is merged from the impressions and conversions data files. The merged data is saved to the data/merged directory. Then we clean the data (remove duplicates, fill missing values, parse user agent strings, etc.) and save the cleaned data to the data/cleaned directory.

- **Impressions:** 1,103,345,895 (1.1 billion)
- **Conversions:** 11,088,921 (11.1 million)
- **Campaigns:** 88

in the training set. The validation and test sets are each 10% of the total merged impressions and conversions data. The columns at this point are described in Table 3. At this point we have

Table 3: DataFrame Structure and Feature Details before preprocessing

#	Column	Dtype	Details
-	unique_id	string	Drop before preprocessor (biases training)
-	impression_dttm_utc	datetime64[ns]	Drop before preprocessor (not needed)
-	device_type	string	Drop (already in ua_* columns)
-	conv_dttm_utc	datetime64[ns]	Drop before preprocessor (not needed)
1	campaign_id	int32	Categorical (Ordinal Encoder)
2	cnxn_type	string	Categorical (Ordinal Encoder)
3	dma	int16	Categorical (Ordinal Encoder)
4	country	string	Categorical (Ordinal Encoder)
5	prizm_premier_code	string	Categorical (Ordinal Encoder)
6	ua_browser	object	Categorical (Ordinal Encoder)
7	ua_os	object	Categorical (Ordinal Encoder)
8	ua_device_family	object	Categorical (Ordinal Encoder)
9	ua_device_brand	object	Categorical (Ordinal Encoder)
10	impression_hour	int8	Cyclical (Numerical Encoder)
11	impression_dayofweek	int8	Cyclical (Numerical Encoder)
12	ua_is_mobile	bool	Boolean (Numerical Encoder)
13	ua_is_tablet	bool	Boolean (Numerical Encoder)
14	ua_is_pc	bool	Boolean (Numerical Encoder)
15	ua_is_bot	bool	Boolean (Numerical Encoder)
16	conversion_flag	int8	*Target*

Once we process the data we save the encoders and scalers to disk to persist. We will see why later.

The processed data roughly coressponds to the cleaned data with one notable change. We encode the time features (hour and day of week) as cyclical features. This is done to ensure that the model can learn the periodic nature of these features. The cyclical encoding is done using the following formula:

$$\text{feature\_name} = \sin\left(\frac{2\pi \cdot \text{feature\_name}}{\text{max\_value}}\right) \text{ and } \text{feature\_name} = \cos\left(\frac{2\pi \cdot \text{feature\_name}}{\text{max\_value}}\right)$$

The processed data shape is described in Table 4. The processed data is saved to the data/processed directory.

Table 4: DataFrame Structure and Feature Details after preprocessing

#	Column	Dtype	Details
1	cat_0	int64	campaign ID
2	cat_1	int64	cnxn type
3	cat_2	int64	dma
4	cat_3	int64	country
5	cat_4	int64	prizm premier code
6	cat_5	int64	ua browser
7	cat_6	int64	ua device brand
8	cat_7	int64	ua device family
9	cat_8	int64	ua os
10	num_0	float64	ua is mobile
11	num_1	float64	ua is tablet
12	num_2	float64	ua is pc
13	num_3	float64	ua is bot
14	num_4	float64	hour sin
15	num_5	float64	hour cos
16	num_6	float64	day sin
17	num_7	float64	day cos
18	conversion_flag	int64	conversion flag

Just as a note the category sizes when we fit the OrdinalEncoder are as follows:

```
 {'campaign_id': 89,
'cnxn_type': 5,
'country': 191,
'dma': 212,
'prizm_premier_code': 70,
'ua_browser': 521,
'ua_device_brand': 74,
'ua_device_family': 7679,
'ua_os': 28}
```

Now that we have the processed data we can move on to training the network. Since the total size of the processed data is around 7GB, we can load the entire dataset into memory. We decided to use a on-demand GPU VM to train the model. Batching the dataset to train on a workstation with 16GB of RAM required dataset batching and sharding. We can implement this in the future if we need to train on larger datasets. However, for now we can skip these steps and keep our implementation simple.

6.3 Impression Stream

[WORK IN PROGRESS] The gist of the implementation is as follows:

- We generate a stream of Impression objects, from the data file using the ImpressionGenerator.
- We preprocess each Impression object using the OnlinePreprocessor so that we can use it for model inference.

- We pass each Impression object to the DecisionLoop, which returns a bid, win/loss, and clearing price.
- We update the ResultsLogger with the results.

6.4 CLI Driver

```
python run_sim.py \
    --data data/clean_data.parquet \
    --model models/best_model.pth \
    --preproc preprocessors/ \
    --out runs/run_01.parquet \
    --num-imps 1_000_000 \
    --device auto
```

7 Open Questions and Future Work

8 Conclusion

AuctioNN provides a controlled environment to test whether modern ML can simultaneously optimise CPA, satisfy campaign obligations, and execute within the harsh latency limits of real-time bidding. The architecture is intentionally minimal yet extensible, enabling rigorous *ablation studies* of every engineering choice—from eligibility gates to neural features to bid shading policies.