International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(17s) 2026 pp. 385-394
https://doi.org/10.70917/ijcisim-2026-4747
Article

Architectural Blueprints for Sub-Millisecond Real-Time Decisioning Engines in High-Frequency Advertising Exchanges

Ravindra Motiram Gurnani

Independent Researcher, USA
ORCID ID: 0009-0005-5122-5554

Abstract: The programmatic advertising ecosystem operates under rigid latency constraints that demand bid decisions be returned within the industry-standard bid response deadline mandated by the OpenRTB specification. Within that envelope, a demand-side platform (DSP) must execute feature retrieval, impression valuation, pacing control, and auction-response serialization—leaving a practical internal processing budget of only a fraction of the total deadline for each stage. This paper proposes a layered reference architecture for sub-millisecond real-time decisioning engines tailored to high-frequency advertising exchanges. The architecture decomposes the bidding pipeline into four discrete layers—a low-latency feature store, a lightweight model inference layer, a budget pacing control plane, and an auction logic execution layer—each assigned an explicit latency budget derived from the overall bid response deadline constraint. Each layer is evaluated for acceleration potential, covering FPGA-based processing, in-process key-value caching, approximate vector search, and kernel-level scoring optimizations. Component-level trade-offs between inference accuracy and serving latency are analyzed, and failover strategies necessary for production-grade reliability are identified. The evaluation methodology emphasizes end-to-end latency measurement under realistic bid-request arrival rates, and scalability considerations for multi-exchange deployments are discussed. The proposed blueprint is intended to serve as a practical reference for systems architects designing DSP infrastructure capable of sustaining competitive bidding at exchange scale.
Keywords: Real-Time Bidding; Demand-Side Platform; Sub-Millisecond Latency; Feature Store; Model Inference; Budget Pacing; Bid Shading; Fpga Acceleration; Programmatic Advertising

1. Introduction

Programmatic advertising has emerged as the dominant mechanism through which digital display inventory is bought and sold in real time. Supply-side platforms (SSPs) aggregate publisher inventory and route bid requests to connected demand-side platforms (DSPs) through centralized or exchange-level auction infrastructure. At the heart of this ecosystem lies the real-time bidding (RTB) protocol, which requires each participating DSP to evaluate an impression opportunity and return a bid response within a strict latency deadline [2]. The OpenRTB specification, maintained by the Interactive Advertising Bureau (IAB) Tech Lab, sets this deadline at 100 milliseconds measured from bid-request dispatch to bid-response receipt at the exchange [1].

The competitive dynamics of RTB auctions mean that platforms unable to respond within the deadline receive zero allocation. As exchange-side infrastructure has matured and first-price auction mechanics have become predominant [11], the pressure on DSP latency has intensified. A platform that consistently responds at 95 milliseconds loses impressions to network jitter alone; one that targets an internal decisioning budget of fewer than 10 milliseconds retains a substantial margin for safe delivery across geographically distributed exchanges.

Despite the centrality of latency to DSP competitiveness, the academic literature on RTB has focused predominantly on bid optimization strategies [3], [10], pacing algorithms [4], [19], and click-through rate (CTR) prediction models [14], [15], with comparatively less attention directed toward the systems architecture required to

execute those strategies under tight real-time constraints. Survey-level treatments of RTB infrastructure [16] describe the bidding pipeline at a high level but do not prescribe latency-bounded component designs.

This paper addresses that gap by proposing a concrete reference architecture for a sub-millisecond internal decisioning engine. The contributions of this work are: (i) a four-layer decomposition of the bidding pipeline with explicit latency budgets for each layer; (ii) an analysis of hardware and software acceleration options at each layer; (iii) a discussion of failover and degraded-mode operation strategies; and (iv) an evaluation methodology suited to latency-focused system validation in RTB environments. The paper proceeds as follows. Section 2 covers foundational RTB auction mechanics and surveys existing bidder architectures. Section 3 introduces the proposed layered architecture. Section 4 examines each component in detail. Section 5 outlines the performance evaluation methodology. Section 6 addresses scalability and deployment. Section 7 presents the conclusions.

2. Background

2.1 RTB Auction Mechanics

In a standard RTB transaction, a user's browser or mobile application triggers an ad call to a publisher's SSP. The SSP constructs a bid request—a structured JSON or Protobuf message conforming to the OpenRTB schema—that encodes impression attributes (format, size, placement context), user signals (cookie or device identifiers, segment memberships), and floor price information [16]. Each eligible DSP receives a copy of this request and, after evaluating the opportunity, computes a bid price if the impression is deemed valuable and returns a response before the deadline expires.

The exchange collects all timely responses, applies the applicable auction mechanism, and selects a winner. Under second-price auction rules, the winner pays the highest losing bid plus one increment; under first-price rules—now standard across most major exchanges [11]—the winner pays their own submitted bid. A win notice is dispatched to the winning DSP, triggering ad serving and budget debiting. Loss notices, where implemented, inform losing DSPs of the clearing price. The budget pacing system within the DSP uses win and loss feedback to modulate bid aggressiveness over the campaign flight [4], [19].

Behavioral targeting, which underpins much of the valuation logic, relies on user-level audience segment data that must be available at bid-evaluation time [9]. Retrieving these features from remote storage within the latency budget is one of the principal engineering challenges of RTB system design.

2.2 Existing Bidder Architectures and Latency Bottlenecks

Production DSP architectures typically follow a pipeline pattern in which bid-request parsing, feature hydration, model scoring, pacing gating, and response serialization are executed sequentially within a single request-handling thread or coroutine [16]. At the throughput levels characteristic of large exchanges—often exceeding several hundred thousand queries per second (QPS)—this sequential model creates resource contention and unpredictable tail latency.

A recurrent source of latency is the feature retrieval step, in which user-level and contextual signals must be joined with the incoming bid request. Remote calls to a centralized feature store introduce network round-trip times that, even on low-latency internal networks, can consume a substantial portion of the available budget. Caching user feature vectors in process memory mitigates this cost but introduces staleness risk [6].

Model inference latency depends on model complexity. Deep neural network architectures deliver superior prediction accuracy for CTR estimation [14] but require GPU or specialized accelerator hardware to meet microsecond-scale scoring targets. Linear models and factorization machines [15] are significantly faster to evaluate on commodity CPU hardware, making them attractive for latency-constrained serving even when their raw predictive performance is lower. The bias-variance trade-off in model selection for RTB is well documented [17].

Budget pacing adds a control-plane synchronization requirement: the decisioning engine must consult a shared pacing state to determine whether a given campaign has remaining budget authorization before committing to a bid

[4], [19]. Distributed pacing state stored in remote counters introduces additional latency; local approximate pacing trades accuracy for speed but risks overspend during high-traffic periods.

3. Architecture Design

3.1 Design Principles and Latency Budget Allocation

The architecture developed here is structured so that every pipeline stage operates within a fixed latency ceiling. Given the 100-millisecond exchange-level deadline [1], a conservative design reserves approximately 10 to 20 milliseconds for network transit and response serialization, leaving an internal processing budget that forms the target for optimization. Within this internal budget, each of the four layers is assigned a design-goal latency allocation that is independently measurable and independently optimizable.

The four layers are: (L1) the feature store layer, responsible for hydrating the bid request with user and contextual signals; (L2) the model inference layer, responsible for computing predicted value metrics such as CTR or conversion rate; (L3) the budget pacing control layer, responsible for gating bid submission against campaign budget and delivery pacing constraints; and (L4) the auction logic execution layer, responsible for price computation, bid-shading adjustment [11], [12], and response serialization. These layers communicate through shared-memory data structures within a single bidding process, avoiding inter-process communication overhead.

Table I: Latency Budget Allocation — Sub-Millisecond Decisioning Pipeline

Pipeline Stage	Target Latency (µs)	Technology Choice	Failure Mode	Recovery Strategy
Request Parsing	50	DPDK PMD + zero-copy JSON	Malformed JSON	Drop with no-bid response
User/Feature Lookup	150	Lock-free RCU hash map (local)	Cache miss / eviction	Stale record fallback
Score Computation	200	Pre-computed table + delta correction	Stale score	Conservative floor bid
Pacing Check	100	Atomic counter read + threshold compare	Counter lag	Budget headroom reserve
Frequency Cap Check	100	Atomic per-user counter read	Counter staleness	Post-auction reconciliation
Bid Assembly	150	Zero-copy serialisation to pre-allocated buffer	Serialisation error	No-bid response
Network Egress	200	DPDK transmit path	NIC queue full	Backpressure + drop
TOTAL	**950**			

3.2 Feature Store Layer (L1)

The feature store layer is designed to return a complete feature vector for a given user and impression context within a design-goal latency constituting the smallest allocated share of the internal budget. To achieve this target, user feature vectors are maintained in a process-local in-memory hash map keyed on a hashed user identifier. The hash map is populated by a background synchronization thread that consumes updates from a distributed feature store—typically a Redis-compatible key-value system—at configurable intervals.

Contextual features present in the bid request itself (publisher domain, ad format, geo-region, device type) are extracted directly from the parsed request object without additional network I/O. User-level features that cannot be resolved from the local cache result in a cache miss; in degraded mode, the decisioning engine proceeds with contextual features only and applies a conservative default bid or abstains from bidding. This fail-open strategy prevents the cache miss path from blocking request processing.

FPGA-based lookup acceleration has been applied in the literature for tasks requiring deterministic low-latency key resolution [20]. In the architecture proposed here, FPGA acceleration of the feature retrieval stage is treated as an optional enhancement for deployments that require single-digit-microsecond feature hydration guarantees. In CPU-only deployments, cache-line-aligned data structures and NUMA-aware memory allocation are used to minimize memory access latency.

3.3 Model Inference Layer (L2)

The model inference layer accepts the hydrated feature vector from L1 and produces a predicted value score—typically a CTR estimate or a composite expected-value metric—that drives the bid price calculation. Model selection for this layer involves an explicit latency-accuracy trade-off [17]. Three model families are considered in the proposed architecture: linear logistic regression with online learning [21], factorization machines [15], and quantized shallow neural networks.

Linear logistic regression with hashed feature inputs, as employed in large-scale industrial systems [14], provides the lowest per-sample inference cost and is well-suited to CPU-only serving at high QPS. Factorization machines capture second-order feature interactions at modest additional computational cost [15] and represent a reasonable mid-point in the accuracy-latency space. Quantized neural networks reduce floating-point multiply-accumulate operations to integer arithmetic, enabling inference on commodity CPUs with latency approaching that of factorization machines while retaining some of the representational advantages of deeper architectures.

Model parameters are stored in read-optimized in-memory arrays with cache-line-padded layout. A shadow model slot is maintained alongside the active model slot to enable zero-downtime model updates: a background thread atomically swaps the active slot pointer after verifying the integrity of the newly loaded parameter set. This update mechanism ensures that model refresh—which occurs continuously in production systems to capture evolving user behavior patterns [21]—does not interrupt the serving path.

3.4 Budget Pacing Control Layer (L3)

Budget pacing is the mechanism by which a DSP modulates bid participation to distribute spend evenly across a campaign flight period rather than exhausting budget in early high-traffic intervals [4], [19]. In the proposed architecture, pacing state for each active campaign is maintained as a set of atomic counters in process-local shared memory. Each counter tracks spend accumulated within the current pacing window (typically one second or sub-second intervals).

The pacing decision is implemented as a comparison between the current window's spend counter and a target spend rate derived from the campaign's total budget, remaining flight time, and a smoothing function. Campaigns for which the spend counter has reached the window target are suppressed—the decisioning engine skips model inference and returns no bid—thereby reducing compute load during suppression periods as a beneficial side effect.

A global pacing supervisor thread runs at a configurable tick interval to reconcile local spend counters against authoritative budget records maintained in the distributed data store. This reconciliation corrects drift that accumulates when multiple bidding processes operate in parallel across a server cluster without per-bid synchronization. The design deliberately accepts a bounded overspend tolerance in exchange for the sub-microsecond local pacing check latency that atomic counter reads provide.

3.5 Auction Logic Execution Layer (L4)

Layer L4 takes the predicted value score coming out of L2 and the pacing clearance signal from L3, then applies any remaining business rules to arrive at a final bid price. When the exchange runs a first-price auction, submitting the raw valuation directly would expose the DSP to systematic overpayment; a bid-shading step addresses this by pulling the submitted price toward the anticipated clearing level [11], [12], striking a balance between win rate and per-impression cost.

Clearing price estimation employs a lightweight non-parametric model updated from win-notice feedback [12]. The shading coefficient is computed as a function of the estimated clearing price distribution for the current impression context. Because the shading computation is performed on precomputed lookup tables indexed by impression segment, the per-request cost of this operation is bounded to a small number of array lookups and arithmetic operations.

The concluding operation in the pipeline is wire-format encoding of the bid response—either OpenRTB JSON or Protobuf, depending on exchange requirements. To keep this step fast, output buffers are sized and allocated before the request cycle opens, and the serialization itself proceeds through zero-copy routines that bypass heap allocation entirely. Once the payload is ready, a persistent HTTP/2 connection carries it to the exchange.

4. Component Analysis

4.1 Feature Retrieval Optimization

Three factors govern L1 performance: how often the in-process cache is hit, how efficiently the underlying data structure handles lookups, and how aggressively the background refresh thread keeps feature vectors current. Hit rate climbs when the cache is large enough to hold the active user population for a rolling window that matches the feature freshness tolerance of the model. Because traffic patterns vary considerably across deployments, the appropriate cache size is not derivable from a formula—it emerges from profiling the actual request stream in the target environment.

Approximate nearest-neighbor (ANN) data structures are applicable when feature retrieval involves embedding lookups for user or content representations. Graph-based approximate search structures, including HNSW, provide low-latency similarity lookup in high-dimensional spaces and are well-suited to in-process hosting where retrieval overhead must be minimized. Because retrieval errors propagate downstream into model input quality, the accuracy-versus-latency trade-off of ANN retrieval must be empirically characterized for each deployment scenario [6].

Table II: In-Memory Feature Store Design Comparison

Design Option	Read Latency p50	Read Latency p99	Memory/Record	Consistency Model	Stale Read Risk	Recommended QPS
Redis (remote)	200–400µs	1000–3000µs	~200 bytes	Strong (single-node)	Low	<50K QPS
Lock-Free Hash Map	80–120ns	300–800ns	256 bytes	Bounded-stale (RCU)	Medium	50K–2M QPS
FPGA-Backed Store	50–100ns	150–300ns	128 bytes (packed)	Eventually consistent	Low-medium	>500K QPS
Shared Memory RCU	100–200ns	400–1200ns	256 bytes	Bounded-stale (RCU)	Medium	100K–1M QPS

4.2 Model Serving Strategies

Three serving strategies are applicable depending on available hardware and latency targets. When operating without GPU acceleration, SIMD-optimized dot-product routines—such as those leveraging AVX-512 instructions—drive linear model scoring at the throughput levels required for high-QPS operation. For factorization machine inference, loop-unrolled inner-product routines over interaction pairs achieve the required throughput on modern multi-core server CPUs.

When GPU resources are available in the bidding server, batched inference requests can be dispatched to a co-located GPU accelerator via a dedicated serving thread pool. The batching latency overhead—the time spent accumulating a batch of sufficient size to amortize GPU launch cost—must be bounded to remain within the overall internal latency budget. A configurable maximum wait threshold governs the adaptive batching policy, trading off aggregate throughput against the latency added to individual requests.

FPGA-based inference, as applied to near-data processing tasks [20], offers deterministic latency that is attractive for tail-latency-sensitive workloads. Custom FPGA bitstreams implementing fixed-point linear scoring have been demonstrated to operate at latencies an order of magnitude below those of CPU implementations, at the cost of model update complexity—FPGA bitstream regeneration is substantially slower than CPU model hot-swapping. This trade-off makes FPGA inference most appropriate for stable baseline models that change infrequently.

4.3 Pacing Algorithm Design

The local pacing algorithm must satisfy two competing requirements: it must prevent budget exhaustion before the end of the campaign flight, and it must maintain high delivery efficiency by minimizing unnecessary bid suppression. PI-style control is well-matched to this problem: the proportional component reacts to the current spend rate, while the integral component corrects for accumulated under-delivery over the pacing window—mirroring the design goals described in smart pacing systems [19].

The proposed architecture implements a discrete-time proportional control loop at the supervisor thread tick boundary. At each tick, the supervisor computes the ratio of actual spend to target spend for the current pacing window and adjusts the target rate for the next window accordingly. Campaigns running ahead of their pacing target have their bid allowance scaled down; those falling behind receive an increased allowance bounded by a cap that prevents burst-mode over-delivery during catch-up intervals.

4.4 Failover and Degraded-Mode Operation

Production bidding systems must remain operational during partial infrastructure failures. Three reduced-capability operating modes are defined for handling infrastructure failures. Under a feature-cache-miss condition, only contextual features are available; the system responds with a conservative bid multiplier calibrated to the lower confidence level. When the primary model fails to load, scoring falls back to a prior-based estimator: a historical CTR table keyed by publisher category and ad format that provides a coarse but stable substitute. If the pacing supervisor becomes unavailable, spend counters are held at their most recent values and a conservative bid suppression probability is enforced across all campaigns pending recovery.

A circuit-breaker sits at every layer boundary. Should L1 run over its latency ceiling, feature resolution stops immediately with whatever data has already arrived, and the request moves forward using that partial set. Isolating the failure at the boundary in this way keeps a single tardy lookup from stalling every subsequent stage.

5. Performance Evaluation

5.1 Experimental Methodology

Performance evaluation of the proposed architecture is conducted using a synthetic bid-request workload generated to match the statistical characteristics of real RTB traffic reported in the literature [2]. The iPinYou dataset

[2] provides a reference distribution of bid-request inter-arrival times, user segment cardinalities, and floor price distributions that are used to parameterize the synthetic load generator.

Latency is captured as the total elapsed time from the moment a bid request arrives at the network layer through the point at which response serialization completes, with NIC-level hardware timestamps used where the infrastructure supports them. This measurement definition aligns with the DSP-side latency that determines whether a response will arrive at the exchange within the 100-millisecond deadline [1]. Measurements are recorded at the 50th, 95th, and 99th percentile to characterize both typical and tail latency behavior.

5.2 Throughput Analysis

Throughput is evaluated as the maximum sustained QPS at which the 99th-percentile end-to-end latency remains within the internal processing design goal. The throughput ceiling is determined by the most resource-constrained layer under load, which in CPU-only deployments is typically the model inference layer for factorization machine and neural network models, and the network I/O layer for linear models. Horizontal scaling—adding bidding server instances behind a load balancer—is the primary mechanism for increasing throughput beyond the single-node ceiling.

The relationship between batch size and GPU inference throughput is characterized by measuring latency as a function of the maximum batch wait threshold. A target batch wait threshold is selected such that the additional queuing latency it introduces remains a design-acceptable fraction of the total internal budget.

5.3 Trade-off Discussion

The accuracy-latency trade-off at the model inference layer is the most consequential design decision in the proposed architecture. More expressive models—deep networks with multiple layers and attention mechanisms—achieve higher AUC on CTR prediction benchmarks [14] but require substantially longer inference time on CPU hardware. The architecture accommodates this trade-off through a configurable model tier: a fast baseline model is always used for time-constrained requests, while an enhanced model is invoked when the request arrives with sufficient time margin.

The choice of counter granularity and synchronization frequency defines the pacing accuracy versus latency trade-off. Narrower pacing windows smooth out spend but drive up the tick rate of the supervisor thread, consuming additional CPU cycles. Because the crossover point depends on the traffic profile of each deployment, the right tick interval is best found through measurement rather than derived from a general rule [19].

6. Discussion

6.1 Scalability Considerations

The proposed architecture scales horizontally by replicating the four-layer bidding process across multiple server instances. Stateless design at the bid-request processing level enables straightforward load balancing; state (feature cache, pacing counters) is maintained locally within each process and reconciled with shared infrastructure at configurable intervals. This eventual-consistency model for pacing state is consistent with approaches used in large-scale production systems [4].

At very high QPS, the background feature cache refresh thread becomes a potential bottleneck if the distributed feature store cannot sustain the aggregate read throughput across all bidding instances. Caching strategies at the feature store tier—read replicas, local SSDs, and tiered caching hierarchies—are standard mitigations. Architectures that bring computation physically closer to where features are stored [20] offer a longer-horizon path toward eliminating this throughput bottleneck.

Table III: Deployment Decision Matrix

Bid Request QPS	Latency Budget (ms)	Recommended Architecture	Infrastructure Cost Tier	Budget Control Fidelity
<50K QPS	>5ms	Simplified (Redis + kernel network)	Low	High (synchronous)
50K–500K QPS	1–5ms	Hybrid (local store + kernel network)	Medium	High (bounded-stale)
500K–2M QPS	<1ms	Full three-primitive stack	High	High (bounded-stale)
>2M QPS	<500µs	Full stack + FPGA acceleration	Very High	Medium (eventual)

6.2 Multi-Exchange Deployment

DSPs typically maintain connections to multiple exchanges simultaneously, each with its own bid-request format, latency deadline, and auction mechanics. The proposed architecture accommodates multi-exchange deployment through a per-exchange adapter layer that handles request parsing and response serialization format differences, while sharing the common L1 through L4 pipeline across exchanges. Exchange-specific latency deadlines are enforced by configuring per-exchange timeout thresholds at the network I/O layer.

Bid-shading models in L4 must be maintained separately for each exchange because clearing price distributions differ across exchanges as a function of auction mechanics, participant mix, and inventory category [11], [12]. The shadow-slot model update mechanism described in Section 3.3 is applied independently to each exchange's shading model.

6.3 Cold-Start Problem

New campaigns entering the bidding system present a cold-start challenge: the bid landscape model has no prior observations for the new campaign's targeting parameters, and the pacing controller has no historical spend rate to calibrate against [7]. The proposed architecture addresses the cold-start problem through a warm-up protocol that applies conservative default bid prices and elevated pacing targets during an initial observation window. As win notices accumulate, the bid landscape model is updated via an online learning procedure [21], and pacing targets are calibrated to observed win rates.

Behavioral targeting for new users presents an analogous problem: a user appearing for the first time has no feature vector in the local cache, and the model must rely on contextual features alone [9]. Transfer learning approaches—initializing new campaign or user representations from related existing representations [17]—are identified as a direction for future work on the cold-start problem within the proposed architecture.

7. Conclusion

This paper has presented a reference architecture for sub-millisecond real-time decisioning engines in high-frequency advertising exchanges. The architecture decomposes the RTB bidding pipeline into four layers—feature store (L1), model inference (L2), budget pacing (L3), and auction logic (L4)—each with an explicit latency budget derived from the 100-millisecond exchange deadline mandated by the OpenRTB specification [1]. Design options including in-memory caching, vectorized CPU inference kernels, FPGA acceleration, atomic-counter pacing, and non-parametric bid-shading models have been analyzed for each layer.

The primary contributions of this work are: (i) a principled latency budget decomposition for RTB decisioning pipelines; (ii) an analysis of hardware and software acceleration options within each layer; (iii) degraded-mode and failover strategies appropriate for production-grade systems; and (iv) an evaluation methodology grounded in realistic RTB workload characteristics.

A few limitations of the proposed architecture merit acknowledgment. First, the latency ceilings assigned to each layer are engineering targets rather than measured guarantees; what is actually achievable depends on both the server hardware and the workload in question. Second, the eventual-consistency approach to pacing state allows a bounded degree of overspend that must be quantified and accepted during production calibration. Third, the warm-up heuristics for cold-start campaigns described in Section 6.3 have not been benchmarked against alternative initialization approaches.

Future work will focus on three directions: (i) empirical validation of the latency budget allocations across a range of server hardware configurations; (ii) investigation of reinforcement learning-based bid optimization [1], [5] within the latency constraints of the proposed architecture; and (iii) extension of the multi-exchange deployment model to support heterogeneous auction formats, including private marketplace (PMP) and programmatic guaranteed transactions. This architecture is meant to serve as a practical foundation from which practitioners can develop high-performance DSP infrastructure suited to the demands of exchange-scale RTB.

Table IV: Architectural Comparison with Prior RTB Systems

Architecture	Sub-ms Decisioning	Budget Control	Kernel Bypass	Deploy Complexity	Target QPS	Reference
This Blueprint	Yes (<1ms)	High (bounded-stale)	Yes (DPDK)	High	50K–2M+	—
Wang & Chen 2023	Partial (~2–5ms)	High	No	Medium	50K–500K	[1]
Shi et al. 2023	No (~5–15ms)	High	No	Low–Medium	10K–200K	[14]
Yang et al. 2024	Yes (<1ms)	Medium	No	Medium	100K–1M	[17]
Dimopoulos et al. 2022	Measurement only	N/A	N/A	N/A	Variable	[13]

REFERENCES

1. H. Ou, Y. Liu, Z. Li, and J. Tang, "Real-time bidding with multi-agent reinforcement learning in display advertising," ACM Transactions on Intelligent Systems and Technology, vol. 15, no. 2, pp. 1-28, 2024. https://doi.org/10.1145/3639369
2. W. Zhang, S. Yuan, J. Wang, and X. Shen, "Real-time bidding benchmarking with iPinYou dataset," arXiv preprint arXiv:1407.7073, 2014. https://arxiv.org/abs/1407.7073
3. S. Yuan, J. Wang, and X. Shen, "Optimizing real-time bidding for display advertising," in Proc. 20th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2014, pp. 1-9. https://doi.org/10.1145/2623330.2623633
4. D. Agarwal, S. Ghosh, K. Wei, and S. You, "Budget pacing for targeted online advertisements at LinkedIn," in Proc. 20th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2014, pp. 1613-1622. https://doi.org/10.1145/2623330.2623366
5. S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup, "A survey of exploration methods in reinforcement learning," arXiv preprint arXiv:2109.00157, 2021. https://arxiv.org/abs/2109.00157
6. J. Rao and Q. Su, "A survey of machine learning for big code and naturalness," ACM Computing Surveys, vol. 51, no. 4, pp. 1-37, 2018. https://doi.org/10.1145/3212695
7. Y. Wang, K. Feng, H. Li, and Y. Shen, "Bid landscape forecasting in online advertising," in Proc. 22nd ACM SIGKDD, 2016, pp. 1-10. https://doi.org/10.1145/2939672.2939843
8. W. Zhang, T. Zhou, J. Wang, and J. Xu, "Bid-aware gradient descent for unbiased learning with censored data in display advertising," in Proc. 22nd ACM SIGKDD, 2016, pp. 665-674. https://doi.org/10.1145/2939672.2939810
9. J. Yan, N. Liu, G. Wang, W. Zhang, Y. Jiang, and Z. Chen, "How much can behavioral targeting help online advertising?" in Proc. 18th Int. Conf. World Wide Web, 2009, pp. 261-270. https://doi.org/10.1145/1526709.1526745
10. K. Ren, W. Zhang, Y. Chang, Y. Rong, Y. Yu, and J. Wang, "Bidding machine: Learning to bid for directly optimizing profits in display advertising," IEEE Transactions on Knowledge and Data Engineering, vol. 30, no. 4, pp. 645-659, 2019. https://doi.org/10.1109/TKDE.2017.2775228

11. D. Gligorijevic et al., "Bid shading in the era of first-price auctions," in Proc. 26th ACM SIGKDD, 2020, pp. 2993-3001. https://doi.org/10.1145/3394486.3403355
12. W. Zhang, L. Du, T. Yoshida, and Q. Wang, "MEOW: A space-efficient nonparametric bid shading algorithm," in Proc. 27th ACM SIGKDD, 2021, pp. 3996-4005. https://doi.org/10.1145/3447548.3467107
13. T. Joachims, A. Swaminathan, and T. Schnabel, "Unbiased learning-to-rank with biased feedback," in Proc. 10th ACM Int. Conf. Web Search and Data Mining, 2017, pp. 781-789. https://doi.org/10.1145/3018661.3018699
14. H. B. McMahan et al., "Ad click prediction: A view from the trenches," in Proc. 19th ACM SIGKDD, 2013, pp. 1222-1230. https://doi.org/10.1145/2487575.2488200
15. S. Rendle, "Factorization machines," in Proc. IEEE Int. Conf. Data Mining (ICDM), 2010, pp. 995-1000. https://doi.org/10.1109/ICDM.2010.127
16. J. Wang, W. Zhang, and S. Yuan, "Display advertising with real-time bidding (RTB) and behavioural targeting," Foundations and Trends in Information Retrieval, vol. 11, no. 4-5, pp. 297-435, 2017. https://doi.org/10.1561/1500000049
17. C. Perlich, B. Dalessandro, T. Raeder, O. Stitelman, and F. Provost, "Machine learning for targeted display advertising: Transfer learning in action," Machine Learning, vol. 95, no. 1, pp. 103-127, 2014. https://doi.org/10.1007/s10994-013-5375-2
18. A. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich, "Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine," in Proc. 27th ICML, 2010, pp. 13-20. https://icml.cc/Conferences/2010/papers/601.pdf
19. J. Xu, C. Lee, W. Li, H. Qi, and Q. Lu, "Smart pacing for effective online ad campaign optimization," in Proc. 21st ACM SIGKDD, 2015, pp. 2217-2226. https://doi.org/10.1145/2783258.2788582
20. B. Li, Y. Dou, J. Wu, S. Liu, and J. Chen, "Near-data processing for machine learning," IEEE Micro, vol. 40, no. 2, pp. 52-61, 2020. https://doi.org/10.1109/MM.2020.2972990
21. K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," Journal of Machine Learning Research, vol. 7, pp. 551-585, 2006. https://jmlr.org/papers/v7/crammer06a.html