INFERENCE AUCTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

When inference demand exceeds available compute capacity, model providers must decide which requests should be served first. Users have different tolerances for delay from an LLM API, but current priority pricing schemes compress these differences into coarse fixed-price service tiers. We design an *inference auction* that allows users to bid for faster service. Our auction allocates priority in an economically efficient way without sacrificing latency, and we develop fast algorithms for implementing prices that incentivize truthful bidding. We also design an autobidding agent for our inference auction, where users specify a budget on total spend and the autobidder dynamically adjusts its bids over time to maximize user utility subject to the budget constraint. Experiments validate the practicality of our auction: it increases system welfare while maintaining the cache utilization and latency advantages of SGLang, a state-of-the-art inference serving framework.

1 INTRODUCTION

Large-scale LLM inference requires allocation of shared compute resources across different users' requests. During periods of heavy traffic, model providers need to determine which requests to serve first, and how much to charge for faster service. Users can differ substantially in their willingness to pay for earlier service. For example, an interactive request may be sensitive to delays that are inconsequential for a background job. Today, most major model providers allocate priority through a small number of fixed-price service tiers (typically two or three), with higher-paying requests receiving priority over lower-paying ones during times of high traffic (e.g., Anthropic, 2026; Amazon Web Services, 2026; OpenAI, 2026). These coarse tiers give users limited scope to express their latency preferences, and as a result, priority is allocated in an economically inefficient way.

The design of a more expressive and efficient market for inference priority raises new challenges that are absent from conventional queuing systems and transaction-fee markets. Modern inference engines process prompts with the same prefix by reusing the key-value (KV) cache across their shared prefix, so the time it takes to serve a request depends on which requests have already been served (e.g., Zheng et al., 2024; Kwon et al., 2023). For example, interrupting a batch of requests that all share a long common prefix to serve unrelated requests may cause the common prefix to be evicted from the KV cache, disrupting cache reuse and harming request throughput. Serving that same batch of common-prefix requests contiguously instead will generally yield improved throughput due to better cache utilization. This form of *prefix-aware* scheduling and KV cache management is implemented in state-of-the-art inference serving frameworks such as SGLang (Zheng et al., 2024), leading to significant latency and throughput gains. A good market design should not sacrifice these performance gains.

We design an *inference auction* that allows users to bid for priority. Our key observation is that preserving the cache-reuse benefits of prefix-aware serving systems still leaves freedom to decide which groups of requests receive priority, enabling significant gains in social welfare while retaining the cache optimality of practical serving systems. Our key contributions, in more detail, are as follows:

Inference Auction Design (Section 2 and Section 3). We develop the first inference auction, a way for LLM providers to prioritize requests by letting users bid for faster service. Each bid represents the user's reported value for moving a request from the back of the scheduling queue to the front, with value increasing linearly as the request moves earlier in the schedule. Our goal is to maximize social welfare—the total value users derive from their assigned priorities—while retaining cache optimality. A key aspect of our inference auction design is to constrain the resulting schedule of

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

requests to be a *depth-first-search (DFS) schedule*, that is, it must correspond to a DFS traversal of the *radix tree* of requests (a tree in which requests that share a prompt prefix share a path from the root). Inference auctions corresponding to such DFS schedules maximize KV cache hit rate due to optimal prefix sharing. We show that the welfare-maximizing DFS schedule corresponds to the DFS traversal that orders a node's children by average bid across requests in their subtrees, and can therefore be computed very efficiently. We pair our scheduling rule with Vickrey–Clarke–Groves (VCG) payments that incentivize truthful bidding. Computing VCG payments naively requires recomputing an optimal schedule for each user, causing a quadratic worst-case dependence on the number of requests. We develop an algorithm that computes all payments in quasilinear run-time by decomposing each user's welfare externality into local changes along the radix tree.

Autobidding Agents for Inference Auctions (Section 4). Inference users typically submit many requests over time and must manage their inference budget. Even when truthful reporting is optimal in each auction in isolation, users still need to choose how aggressively to bid across requests so that their inference budget is spent where it is most valuable. To address this, we design an autobidding agent that adjusts a user's bids over time to maximize their cumulative utility, subject to a budget constraint. Our autobidding algorithm scales each request's value by a multiplier and adjusts this multiplier over time using gradient descent. We show that this strategy achieves $O(\sqrt{T})$ regret relative to the optimal-in-hindsight bidding policy on a sequence of T inference auctions. We also characterize the welfare loss when users have different pacing multipliers: we show that welfare only decreases when these differences reverse the ordering of requests in the DFS schedule, and we bound the resulting inefficiency by the variation in pacing multipliers across the radix subtrees.

Experiments (Section 5). We implement our inference auction as a scheduling layer in front of an unmodified SGLang server and evaluate it across several workloads with various shared prefix structures. Compared to a bid-blind, cache-optimal request ordering, our mechanism improves overall welfare by $12 - 18.6\%$ under i.i.d. request valuations and up to 100% when valuations are non-i.i.d., all while leaving cache hit rate, average time-to-first-token, and makespan virtually unchanged. Simultaneously, we find that our mechanism enjoys $\sim 80\%$ of the welfare of the unconstrained schedule ordered by decreasing bids. In contrast, this unconstrained schedule sharply reduces cache hit rates, which results in an increase in average latency by as much as $12\times$.

2 Cache-Optimal Schedules and Our Bidding Environment

Serving an autoregressive language model has two stages. During *prefill*, the model processes a request's prompt and stores its keys and values in the KV cache. During *decoding*, it generates output tokens while reusing the cached keys and values. Modern inference engines interleave the two stages across requests, typically prioritizing the decode steps of requests already in progress. Our mechanism determines the order in which queued requests are admitted for prefill.

The KV cache entries for a prompt prefix can be reused by other requests which share that prefix, reducing prefill computation. Because cache capacity is limited, this reuse depends on the request scheduling order: processing requests with a long shared prefix consecutively can avoid evicting and recomputing their common entries. The inference engine SGLang (Zheng et al., 2024) puts this principle into practice by storing shared token prefixes in a radix tree and scheduling requests in the order of a DFS traversal of that tree, which is a longest-prefix-match order and is cache-optimal. *A key aspect of our auction design is to allow bids to only affect the ordering of children in a DFS traversal of the request radix tree.* This ensures that no matter the precise economics of the auction (e.g., schedules, payments, incentives), we maintain the cache efficiency of prefix-aware scheduling. We now formally introduce the elements of the auction environment.

Notation. $\mathcal{S}_n$ denotes the set of all permutations of $\{1, \ldots, n\}$. For a set of indexed elements $\mathbf{a}_{1:m} \coloneqq \{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$, we write $\mathbf{a}_{-i} \coloneqq \mathbf{a}_{1:m} \setminus \{\mathbf{a}_i\}$ to denote $\mathbf{a}_{1:m}$ with element i excluded.

Requests and Scheduling Batches. We consider a setting where a model provider is serving requests to a single LLM on one inference worker. Requests arrive asynchronously and are periodically grouped into scheduling batches of n queued requests. We treat the formation of these batches as exogenous: our mechanism determines the relative prefill priority of requests within a batch, but not which requests enter the batch.[1]

[1] In practice, batches would be formed only during times of high traffic when requests start to queue.

Let $\mathcal{R} \coloneqq \{1, \ldots, n\}$ be the requests in a batch. There are $m \leq n$ inference users who may each submit multiple requests. We use $\mathcal{R}_i \subseteq \mathcal{R}$ to denote the set of requests submitted by user i, where $\bigcup_{i=1}^{m} \mathcal{R}_i = \mathcal{R}$ and $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$ for all $i \neq j$. A *schedule* is a permutation $\sigma \in \mathcal{S}_n$, where $\sigma(r)$ is the position of request r within the batch. Smaller positions receive earlier prefill service, and the scheduler admits requests for prefill according to this ordering. Each request is represented by its sequence of prompt tokens. Let M denote the total number of prompt tokens across all requests in $\mathcal{R}$. The shared-prefix structure of $\mathcal{R}$ is captured by a radix tree.

Definition 2.1 (Request Radix Tree)**.** *The radix tree* T *for a set of requests is the rooted, edge-labeled tree obtained by merging common prefixes and contracting every non-branching path into a single edge. Each request corresponds to a leaf, and the lowest common ancestor of two request leaves represents their longest shared prefix. For a node* u*, let* $\mathcal{C}(\mathsf{u})$ *denote its children,* $\mathcal{L}(\mathsf{u})$ *denote the request leaves in the subtree rooted at* u*, and* $N(\mathsf{u}) \coloneqq |\mathcal{L}(\mathsf{u})|$*. The maximum branching factor of* T *is* $\nu \coloneqq \max_{\mathsf{u}} |\mathcal{C}(\mathsf{u})|$*. See Figure 5 for an example.*

Of particular interest are schedules that process all requests in one radix subtree before moving on to another, that is, they correspond to DFS traversals of T.

Definition 2.2 (DFS Schedule)**.** *A schedule* σ *is a DFS schedule if for every node* u*, the leaves belonging to each child subtree* $\mathsf{c} \in \mathcal{C}(\mathsf{u})$ *appear contiguously in* σ*. Let* $\mathcal{D}(\mathsf{T}) \subseteq \mathcal{S}_n$ *denote the set of DFS schedules for* T*.*

Under the assumption that the KV cache is large enough to hold the longest request in the batch, schedules in $\mathcal{D}(\mathsf{T})$ enjoy the optimal cache hit rate by Theorem 3.1 in Zheng et al. (2024).

Values and Priority. The user who submits request r has a private value $v_r \geq 0$ for receiving earlier service. The *fractional priority* received by request r under schedule σ is $x_r(\sigma) \coloneqq \frac{|s \in \mathcal{R} : \sigma(s) > \sigma(r)|}{n-1}$. The first request has fractional priority 1, the last has fractional priority 0, and a request scheduled ahead of an α-fraction of the batch receives priority α. Let $\mathbf{v}_i \coloneqq [v_r]_{r \in \mathcal{R}_i}$ denote the vector of user i's values for his requests and $\mathbf{x}_i(\sigma) \coloneqq [x_r(\sigma)]_{r \in \mathcal{R}_i}$ be the vector of the corresponding fractional priorities under σ. User i's total value under σ, interpreted as his maximum willingness-to-pay for fractional priority $\mathbf{x}_i(\sigma)$, is $V_i(\sigma) \coloneqq \langle \mathbf{v}_i, \mathbf{x}_i(\sigma) \rangle$ and the *welfare* of σ is $W(\sigma; \mathbf{v}_{1:m}) \coloneqq \sum_{i=1}^{m} V_i(\sigma)$.

Fractional priority provides a simple rank-based proxy for queueing delay and the resulting time-to-first-token (TTFT). *Modeling value as being linear in fractional priority is a design choice to keep the bidding language simple*: users only need to report how much they value moving a request from the back of the scheduling batch to the front, rather than specify a potentially complex value function over TTFT. Our experiments show that bidding according to fractional priority leads to schedules that closely track realized TTFT, supporting fractional priority as a practical proxy for latency.

Bids, Schedules, and Payments. Each request r has a corresponding user bid $b_r \geq 0$, interpreted as the user's reported value for r receiving full priority. Let $\mathbf{b}_i \coloneqq [b_r]_{r \in \mathcal{R}_i}$ be the vector of user i's bids. An inference auction is a mechanism which maps bids to a DFS schedule and user payments.

Definition 2.3 (Inference Auction)**.** *Given a radix tree* T*, an inference auction consists of (1) a* scheduling rule $A : \mathbb{R}_+^n \to \mathcal{D}(\mathsf{T})$ *mapping bids to DFS schedules and (2) a* payment rule $p : \mathbb{R}_+^n \to \mathbb{R}_+^m$ *mapping bids to payments, where* $p_i(\mathbf{b}_{1:m})$ *is the payment charged to user* i*.*

Any inference auction enjoys optimal cache utilization by Theorem 3.1 of Zheng et al. (2024). We make the standard assumption that users have quasilinear utility, so a user with true values $\mathbf{v}_i$ obtains utility $V_i(A(\mathbf{b}_{1:m})) - p_i(\mathbf{b}_{1:m})$ in an inference auction with scheduling rule A and payment rule p. Our goal in the sequel is to design an inference auction that maximizes welfare over $\mathcal{D}(\mathsf{T})$ while incentivizing truthful bidding. A simple symmetry argument shows that the welfare loss due to restricting to $\mathcal{D}(\mathsf{T})$ is no more than 50%.[2]

Proposition 2.4. *For any* $\mathbf{v}_{1:m}$*,* $\max_{\sigma \in \mathcal{D}(\mathsf{T})} W(\sigma; \mathbf{v}_{1:m}) \geq \frac{1}{2} \max_{\sigma \in \mathcal{S}_n} W(\sigma; \mathbf{v}_{1:m})$.

[2] At the same time, the welfare-maximizing DFS schedule can improve significantly over an arbitrary schedule in $\mathcal{D}(\mathsf{T})$. As an example, consider the setting where one request has value $v^\star > 0$ and shares no common prefix with the $n-1$ other requests, each of which has value 0 and a common shared prefix. A uniformly random DFS schedule would obtain zero welfare with probability $1/2$, whereas the optimal DFS schedule always achieves the maximum welfare $v^\star$.

ALGORITHM 1: Average-Bid DFS Scheduling Rule $\bar{A}(\mathbf{b}_{1:m})$

Let u be the root of the radix tree T
Initialize $\sigma \leftarrow [\,]$
$\mathrm{DFS}(\mathsf{u}, \sigma)$
return σ

ALGORITHM 2: $\mathrm{DFS}(\mathsf{u}, \sigma)$

if u is a leaf **then**
  $\sigma.\mathtt{append}(\mathsf{u})$
$[\mathsf{c}_1, \ldots, \mathsf{c}_{d(\mathsf{u})}] \leftarrow \mathcal{C}(\mathsf{u})$ sorted in descending order of F_b/N
for $k = 1, \ldots, d(\mathsf{u})$ **do**
  $\mathrm{DFS}(\mathsf{c}_k, \sigma)$

3 Welfare-Optimal Inference Auctions

We now design a welfare-maximizing (or *economically efficient*) inference auction that is simple and tractable. In Section 3.1, we design a scheduling rule that maximizes reported user welfare. The scheduling rule admits a clean algorithm: at every node of the radix tree, visit its children in decreasing order of their average sub-tree bid. In Section 3.2, we implement VCG prices that incentivize truthful bidding. Although these payments are defined through a separate counterfactual optimization for each user, we show that they can be computed in quasilinear total run-time. Taken together, our scheduling and payment rules maximize *true* user welfare and can be implemented with very little overhead compared to the time it takes to construct and traverse the radix tree.[3]

3.1 Implementing Welfare-Maximizing DFS Schedules

Since we restrict to DFS schedules, the scheduler's only decision is how to order the children of each internal node in the radix tree. For a node u, let $F_b(\mathsf{u}) \coloneqq \sum_{r \in \mathcal{L}(\mathsf{u})} b_r$ denote the sum of bids in the subtree rooted at u. We will show that reported welfare is maximized by visiting the children of every internal node in decreasing order of $F_b(\mathsf{u})/N(\mathsf{u})$.

For intuition as to why this is the case, consider two children c and d of the same internal node. If c is visited before d, every request in $\mathcal{L}(\mathsf{c})$ is scheduled ahead of every request in $\mathcal{L}(\mathsf{d})$, and these cross-subtree pairs contribute $\frac{N(\mathsf{d})F_b(\mathsf{c})}{n-1}$ to reported welfare. Visiting d before c instead contributes $\frac{N(\mathsf{c})F_b(\mathsf{d})}{n-1}$. It is therefore welfare-improving to place c before d whenever $N(\mathsf{d})F_b(\mathsf{c}) \geq N(\mathsf{c})F_b(\mathsf{d})$, or equivalently when $F_b(\mathsf{c})/N(\mathsf{c}) \geq F_b(\mathsf{d})/N(\mathsf{d})$. The following theorem shows that applying this local rule independently at every branch produces a globally-optimal DFS schedule.

Theorem 3.1. *Given bid vectors $\mathbf{b}_{1:m}$, the average-bid DFS scheduling rule $\bar{A}(\mathbf{b}_{1:m})$ given by Algorithm 1 returns a cache-optimal schedule σ that maximizes $W(\sigma; \mathbf{b}_{1:m})$ among all DFS schedules. Furthermore, this schedule can be computed in $O(M + n \log \nu)$ expected run-time.*

Figure 6 illustrates an example of an average-bid DFS schedule.

3.2 Computing VCG Payments

The average-bid DFS of Algorithm 1 yields a schedule that maximizes reported welfare, but our ultimate goal is to maximize welfare with respect to users' true values. Vickrey (1961)-Clarke (1971)-Groves (1973) (VCG) payments ensure this property by charging each user for the welfare loss that his bids impose on the other users. When paired with a scheduling rule that maximizes reported welfare over a fixed set of feasible outcomes, these payments make truthful bidding (i.e., bidding $\mathbf{b}_i = \mathbf{v}_i$) an optimal strategy for users with quasilinear utility. We instantiate the VCG mechanism with average-bid DFS schedules and $\mathcal{D}(\mathsf{T})$ as the feasible set:

Definition 3.2 (VCG payment rule)**.** *Let $h_i(\mathbf{b}_{-i}) = \max_{\sigma \in \mathcal{D}(\mathsf{T})} W(\sigma; (\mathbf{0}, \mathbf{b}_{-i}))$ denote the maximum reported welfare the other users could obtain if user i's bids were set to zero while its requests remain in the radix tree.*[4] *The VCG payment rule charges user i*

$$p_i^{\mathtt{VCG}}(\mathbf{b}_{1:m}) = h_i(\mathbf{b}_{-i}) - \sum_{j \neq i} \langle \mathbf{b}_j, \mathbf{x}_j(\bar{A}(\mathbf{b}_{1:m})) \rangle.$$

[3] Inference frameworks such as SGLang already construct and traverse the radix tree to implement prefix-aware scheduling. We only incur an additional $O(\log \nu)$ multiplicative factor.

[4] The VCG counterfactual $h_i(\mathbf{b}_{-i})$ is often described informally as removing user i from the auction. In our setting, removing a user's requests would change the radix tree and hence the set of feasible schedules. In fact, this (incorrect) version of VCG would require the platform to pay some users (see Appendix B). We instead fix the set of requests and assign user i his weakest possible type, the zero bid vector.

The inference auction with average-bid DFS schedules and VCG payments satisfies the following properties:

1. **Efficiency:** Under truthful bidding, true welfare $W(\sigma; \mathbf{v}_{1:m})$ is maximized over DFS schedules.
2. **Incentive compatibility (IC):** $V_i(\bar{A}(\mathbf{v}_i, \mathbf{b}_{-i})) - p_i^{\texttt{VCG}}(\mathbf{v}_i, \mathbf{b}_{-i}) \geq V_i(\bar{A}(\mathbf{b}_i, \mathbf{b}_{-i})) - p_i^{\texttt{VCG}}(\mathbf{b}_i, \mathbf{b}_{-i})$ for all $\mathbf{v}_i, \mathbf{b}_i, \mathbf{b}_{-i}$. This makes truthful bidding an optimal strategy.
3. **Individual rationality (IR):** $V_i(\bar{A}(\mathbf{v}_i, \mathbf{b}_{-i})) - p_i^{\texttt{VCG}}(\mathbf{v}_i, \mathbf{b}_{-i}) \geq 0$ for all $\mathbf{v}_i, \mathbf{b}_{-i}$.
4. **No positive transfers:** $p_i^{\texttt{VCG}}(\mathbf{b}_{1:m}) \geq 0$ for all bid vectors $\mathbf{b}_{1:m}$.

Furthermore by Krishna & Perry (1998), these payments maximize the model provider's revenue among all IC and IR mechanisms that implement the welfare-maximizing DFS schedule.

Efficient Payment Computation. Naively computing VCG payments for each user consists of repeatedly zeroing out that user's bids and rerunning Algorithm 1. This approach takes $O(mn \log \nu)$ time, which is quadratic in n if each user has only a single request. While such a quadratic dependence on n is unavoidable if the goal is to return the welfare-optimal DFS traversal when each user is excluded, we are able to bypass this barrier since we only need the optimal welfare values. As a result, our algorithm enjoys an improved runtime of $O(\min\{M, mn\} \log \nu)$.

The key idea behind our algorithm (Appendix B.2) is easiest to see when each user submits a single request. Suppose we set the bid of request r to zero. At every internal node on the path from the root to r, only the average bid of the child containing r changes. This average can only decrease, and so the affected child can only move later in the traversal while all other children retain their relative order. Siblings that remain on the same side of the affected child do not contribute to the user's payment, while each sibling that overtakes it contributes a local externality. Since the children are already sorted by average bid from the average-bid DFS schedule, a binary search identifies the siblings that overtake the affected child. Summing these local effects over the root-to-leaf path then gives the user's VCG payment.

When a user submits multiple requests, zeroing out all of his bids may affect multiple children of a given internal node. However the same principle still applies: jointly reorder the affected children according to their modified average bids and merge them with the unchanged children.

Theorem 3.3. *Given as input the processed radix tree* T*, where at every node* u *(1)* $F_b(\mathsf{u})$ *and* $N(\mathsf{u})$ *are stored and (2) its children are sorted in decreasing order of average bid, all VCG payments* $p_1^{\texttt{VCG}}(\mathbf{b}_{1:m}), \ldots, p_m^{\texttt{VCG}}(\mathbf{b}_{1:m})$ *can be computed in* $O(\min\{M, mn\} \log \nu)$ *total run-time.*

The end-to-end run-time of implementing the welfare-optimal DFS schedule with VCG prices is therefore $O(M + \min\{M, mn\} \log \nu)$.

4 Autobidding Agents for Inference Auctions

The VCG mechanism makes truthful bidding optimal within any single scheduling batch. However, inference users typically have many requests that need to be served over time, and they may have a budget allocated for the total amount they can spend in a given time period. This budget constraint couples the user's bids, as bidding truthfully in every batch can exhaust the budget too early. Motivated by autobidding systems in online advertising,[5] we develop an autobidding agent for inference auctions: an online policy that converts a user's values and remaining budget into bids, without requiring a model of future demand or competing bids.

Repeated Auction Model. Consider a sequence of T scheduling batches. In batch $t \in [T]$, let $\mathsf{T}^{(t)}$ be the request radix tree, $\mathcal{R}_1^{(t)}$ be the autobidder's requests, $\mathbf{v}_1^{(t)} = [v_r^{(t)}]_{r \in \mathcal{R}_1^{(t)}}$ be his value vector, $\mathbf{b}_1^{(t)}$ be his bid vector, and $\mathbf{b}_{2:m}^{(t)}$ be the other users' bids.[6] We consider the welfare-maximizing inference auction of Section 3. Given bids $\mathbf{b}_{1:m}^{(t)}$, the mechanism selects schedule $\bar{A}^{(t)}(\mathbf{b}_{1:m}^{(t)})$ and charges the autobidder his VCG payment $p_1^{(t)}(\mathbf{b}_{1:m}^{(t)})$. The autobidder's value in batch t is $V_1^{(t)}(\sigma^{(t)}) := \langle \mathbf{v}_1^{(t)}, \mathbf{x}_1^{(t)}(\sigma^{(t)}) \rangle$. The autobidder has total budget $B = T\gamma$ (where $\gamma > 0$ is the

[5] See Section 6 for more background in autobidding in ad auctions.

[6] We use the convention that the autobidder's index is 1, which is without loss of generality.

ALGORITHM 3: Pacing Autobidder for Welfare-Optimal Inference Auctions

Input: Target spend γ, step size $\eta > 0$
Initialize $B^{(1)} = T\gamma$, $\mu^{(1)} = 0$
for $t = 1, \dots, T$ **do**
  Let $\alpha^{(t)} = \frac{1}{1+\mu^{(t)}}$ and bid $\mathbf{b}_1^{(t)} = \alpha^{(t)}\mathbf{v}_1^{(t)}\mathbb{1}\{\|\alpha^{(t)}\mathbf{v}_1^{(t)}\|_1 \le B^{(t)}\}$
  Observe payment $p_1^{(t)}(\mathbf{b}_{1:m}^{(t)})$ and update budget $B^{(t+1)} \leftarrow B^{(t)} - p_1^{(t)}(\mathbf{b}_{1:m}^{(t)})$
  Update $\mu^{(t+1)} \leftarrow \left[\mu^{(t)} + \eta(p_1^{(t)}(\mathbf{b}_{1:m}^{(t)}) - \gamma)\right]_+$

target spend per batch), and seeks to maximize cumulative utility subject to the budget constraint $\sum_{t=1}^{T} p_1^{(t)}(\mathbf{b}_{1:m}^{(t)}) \le T\gamma$. We measure the performance of the autobidder through the notion of regret.

Definition 4.1 (Autobidder Regret)**.** *For a realized sequence of instances* $(\mathbf{v}_1^{(t)}, \mathsf{T}^{(t)}, \mathbf{b}_{2:m}^{(t)})_{t=1}^{T} \sim \mathcal{P}$*, the optimal cumulative utility in hindsight is* $\mathrm{OPT}(T) := \max_{\mathbf{b}_1^{(1)},\dots\mathbf{b}_1^{(T)}} \left\{\sum_{t=1}^{T} V_1^{(t)}(\sigma^{(t)}) - p_1^{(t)}(\mathbf{b}_{1:m}^{(t)}) \;:\; \sum_{t=1}^{T} p_1^{(t)}(\mathbf{b}_{1:m}^{(t)}) \le T\gamma\right\}$.

Regret is the expected difference between $\mathrm{OPT}(T)$ *and the autobidder's cumulative utility; i.e.,* $R(T) := \mathbb{E}\left[\mathrm{OPT}(T) - \sum_{t=1}^{T} V_1(\sigma^{(t)}) - p_1^{(t)}(\mathbf{b}_{1:m})\right]$.

Value Pacing. At every timestep, our autobidder scales the request values by a common pacing factor $\alpha^{(t)}$, bids $\mathbf{b}_1^{(t)} = \alpha^{(t)}\mathbf{v}_1^{(t)}$, and updates the pacing factor $\alpha^{(t)} \to \alpha^{(t+1)}$. The pacing factor decreases when spending exceeds the per-batch target γ, and increases when spending falls below the target. Pseudocode for our autobidding agent is given in Algorithm 3.

Theorem 4.2. *Suppose that* $(\mathbf{v}_1^{(t)}, \mathsf{T}^{(t)}, \mathbf{b}_{2:m}^{(t)})_{t=1}^{T}$ *are drawn i.i.d. from an unknown, finite support distribution* $\mathcal{P}$*. Let* $K := \max_{t\in[T]} |\mathcal{R}_1^{(t)}|$*,* $\|\mathbf{v}_1^{(t)}\|_\infty \le C$*, and suppose that schedules and payments are assigned according to the inference auction of Section 3. Then if* $\eta = \Theta(1/\sqrt{T})$*, Algorithm 3 obtains regret* $R(T) = O(\mathrm{poly}(C, K, \gamma)\sqrt{T})$ *when* T *is sufficiently large.*

The proof proceeds via a reduction to Balseiro et al. (2020), who study online allocation problems with generic concave reward functions and resource constraints. The main technical difficulty is showing that their framework actually applies in our setting. The distributional assumption models a stationary serving environment in which request values, prefix structures, and competing bids can vary across batches but arise from a stable workload. Since bids, values, and prompts are all finite, the assumption that $\mathcal{P}$ has finite support is without loss of generality.

Welfare Under Heterogeneous Pacing. When multiple autobidders pace independently, their multipliers can differ, and so their bids will generally not be proportional to a common pacing factor. This can lead to a loss in per-batch social welfare compared to the welfare under truthful reporting. We now quantify this loss as a function of the request radix tree's structure.

For the remainder of this section, we suppress the batch index and write each bid as $b_r = \alpha_r v_r$. For a subtree c, let $F_v(\mathsf{c}) := \sum_{r\in\mathcal{L}(\mathsf{c})} v_r$ and define its *value-weighted pacing factor* by $\alpha(\mathsf{c}) := F_b(\mathsf{c})/F_v(\mathsf{c})$.

Definition 4.3 (Node Pacing Condition Number)**.** *The condition number of node* u *is* $\kappa_{\mathsf{u}} := \max\{\alpha(\mathsf{c}) : \mathsf{c} \in \mathcal{C}(\mathsf{u}), F_v(\mathsf{c}) > 0\}/\min\{\alpha(\mathsf{c}) : \mathsf{c} \in \mathcal{C}(\mathsf{u}), F_v(\mathsf{c}) > 0\}$*, with* $\kappa_{\mathsf{u}} := 1$ *if at most one child of* u *has* $F_v(\mathsf{c}) > 0$*.*

Condition number measures how unevenly sibling subtrees are paced. When $\kappa_{\mathsf{u}} = 1$, pacing preserves the welfare-maximizing order, whereas larger values of κ_{u} result in greater local distortion. The following theorem formalizes this relationship by lower-bounding the realized welfare in terms of the node-level condition numbers.

Theorem 4.4. *Suppose that each request* r *has corresponding bid* $b_r = \alpha_r v_r$ *for pacing factors* $\alpha_1, \dots, \alpha_n \in (0, 1]$*, and let* Ω_{u} *denote the marginal contribution of node* u *to the optimal DFS welfare. Then the social welfare for the batch is bounded as*

$$\frac{W(\bar{A}(\mathbf{b}_{1:m}); \mathbf{v}_{1:m})}{\max_{\sigma\in\mathcal{D}(T)} W(\sigma; \mathbf{v}_{1:m})} \ge \frac{\sum_{\mathsf{u}} \Omega_{\mathsf{u}}/\kappa_{\mathsf{u}}}{\sum_{\mathsf{u}} \Omega_{\mathsf{u}}} \ge \frac{1}{\max_{\mathsf{u}} \kappa_{\mathsf{u}}}.$$

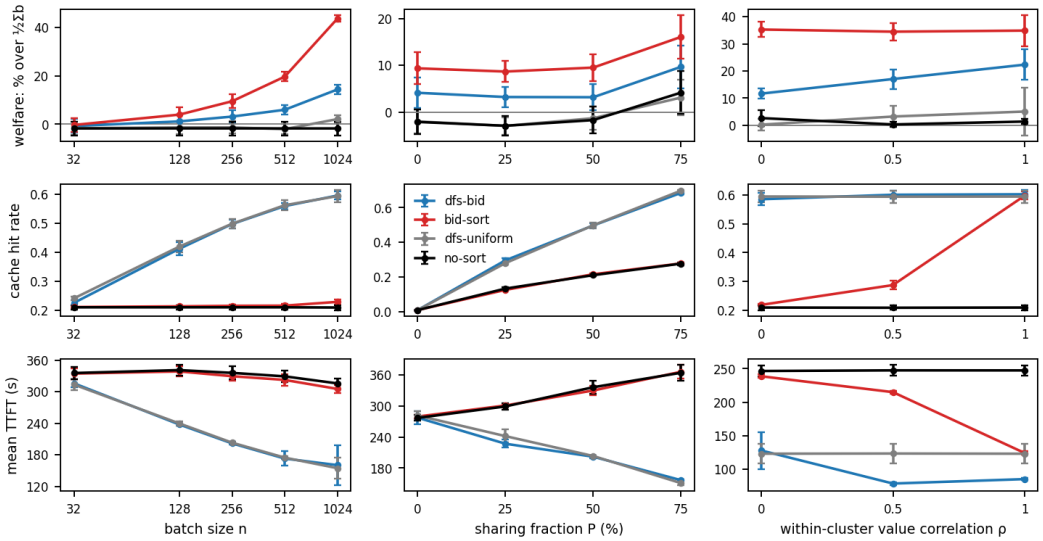


Figure 1: Stars topology. Welfare, cache hit rate, and mean TTFT, varying: batch size n (left), sharing fraction P with n=256 (middle), and within-cluster bid correlation ρ (right).

Theorem 4.4 isolates the economically relevant source of inefficiency from value pacing in inference auctions. Pacing dispersion only matters when it changes the average-bid ordering of sibling subtrees, and the loss in welfare is weighted by $1/\kappa_{\mathrm{u}}$. Heterogeneous request-level pacing may therefore be harmless when it "averages out" within subtrees; what matters more is any systematic pacing variation across sibling subtrees that directly compete for scheduling priority. See Figure 7 for an example.

5 Experiments

Experiment setup. We evaluate our algorithm as a layer on top of the SGLang (Zheng et al., 2024) inference system across several semi-synthetic workloads, on a single 80GB H100 serving a 32B LLM (DeepSeek-AI, 2025). We assume the server is saturated: a waiting queue holds $Q = 1024$ queries, which we split into batches of n before ordering with our algorithm. We configure SGLang to process queries first-come-first-served to preserve their order.

Algorithms. DFS-BID is the average-bid DFS of Algorithm 1 with VCG payments as in Section 3.2. DFS-UNIFORM runs the same DFS with all bids equal (meant to replicate SGLang's longest-prefix-match policy); BID-SORT sorts queries by bid and ignores the prefix tree (it is the unconstrained welfare maximizer in Prop. 2.4); NO-SORT serves queries in arrival order. Values are synthetic, drawn i.i.d. (unless otherwise noted) from $1 + \mathrm{Pareto}(1.5)$, and bids are truthful. Every number reported is a mean of three trials with 1-standard-error bars.

Workloads. We construct semi-synthetic workloads from real LLM evaluation data, covering three basic radix tree topologies: stars, chains, and 3-ary trees. For stars, MIXSHARE mixes SQuAD (Rajpurkar et al., 2016) questions sharing an article as a prefix with GSM8K (Cobbe et al., 2021) questions sharing few-shot examples; in MIXSHARE-P, a $P\%$ of the queries share prefixes and the rest are singletons. For chains we replay SWE-agent (Yang et al., 2024) trajectories. For trees, we run tree-of-thought (Yao et al., 2023) in a short variant and a long variant over articles of 3–5k tokens. Prompts are ≈1k tokens on stars, 3k on chains, and 160 or 4k on the two tree variants, with 60–97% of tokens shared (Appendix D.2). The model's weights leave a 23k-token KV cache; the short trees fit in the cache entirely and serve as a control, while the other workloads force cache eviction.

Results. We measure welfare on the realized order, i.e., with queries ranked by TTFT, and plot it as the percentage above $\frac{1}{2}\sum_r b_r$ (this quantity is the expected welfare of a uniformly random DFS order). We also report the cache hit rate as reported by SGLang and the mean TTFT over the queue.

In every experiment, DFS-BID and DFS-UNIFORM have the same cache hit rate and the same TTFT, but DFS-BID has substantially higher welfare. BID-SORT has higher welfare, but it interleaves

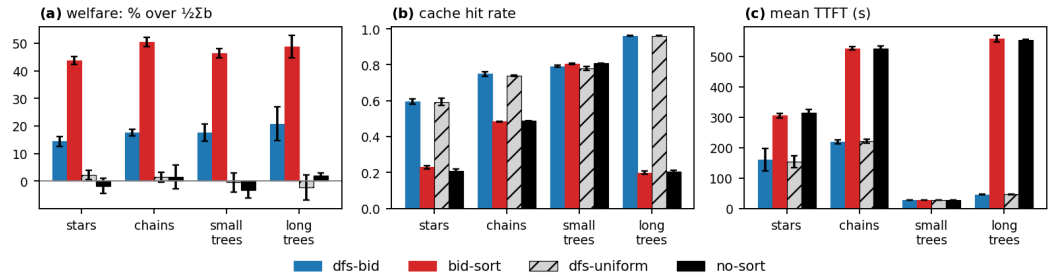


Figure 2: Tree topologies at $n = Q$. When the workload exceeds the cache (all topologies except small trees), BID-SORT pays for its welfare lead (left) in cache misses (middle) and latency (right).

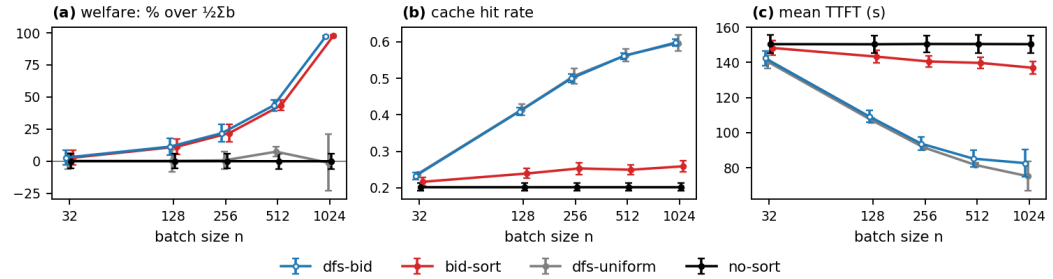


Figure 3: One cluster has non-zero bids, everyone else bids zero (MIXSHARE-50). DFS-BID matches BID-SORT's welfare (left; curves offset slightly for readability) while keeping DFS-UNIFORM's hit rate (middle) and latency (right).

queries that share a prefix, significantly reducing cache hit rate. At $n = Q$, DFS-BID's welfare is within 19–22% of BID-SORT's welfare, well under the 50% that Proposition 2.4 allows.

The first column of Figure 1 varies the batch size on MIXSHARE-50. With more queries in a batch there are more shared prefixes to group, so both DFS orders gain cache hits and get faster with n while BID-SORT and NO-SORT do not. In terms of welfare, DFS-BID's lead over DFS-UNIFORM grows from 1% at $n = 32$, with few shared prefixes per batch, to 12% at $n = Q$. The second column varies the sharing fraction at $n = 256$ with the same result: more sharing helps the DFS orders and hurts the flat ones, and DFS-BID stays 4–7% above DFS-UNIFORM. The third column varies ρ, how much values are correlated within clusters. BID-SORT is substantially slower than DFS-BID until $\rho = 1$, when cluster members are naturally grouped by having the same bid; even then, DFS-BID achieves competitive welfare with better TTFT.

Figure 2 repeats the comparison at $n = Q$ across tree topologies. BID-SORT's TTFT is 2.4× DFS-BID's on chains and 12× on long trees. The short trees are the exception since they fit in the cache, so nothing is ever evicted and every order gets the same hit rate and the same TTFT regardless of how it schedules. This is the one case where the DFS restriction buys nothing and costs 20% of welfare.

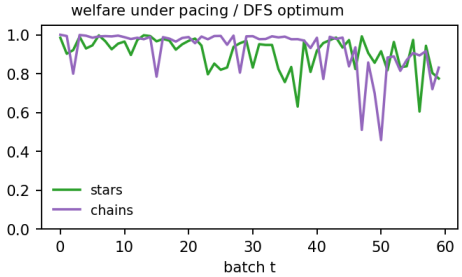


Figure 4: Welfare under autobidding when $T = 60$. Details are in Appendix D.11.

Figure 3 shows the case most favorable to our mechanism (and perhaps the most realistic): one cluster of queries with a shared prefix has non-zero bids, and every other request bids zero. Serving that cluster first is both the unconstrained optimum and a DFS order, so DFS-BID matches BID-SORT's welfare at every batch size while keeping DFS-UNIFORM's hit rate, and BID-SORT pays about 1.5× the TTFT.

Finally, in Figure 4 we simulate the pacing autobidders of Algorithm 3 on the same trees. Autobidding in repeated auctions causes welfare to decrease by about 10% on average compared to the single-shot setting, and welfare is always above what Theorem 4.4 predicts.

6 Related Work

Prefix-Aware LLM Serving. Modern LLM serving frameworks improve request throughput via continuous batching and efficient management of the KV cache. vLLM (Kwon et al., 2023) uses ideas from virtual memory and paging to reduce KV cache fragmentation. SGLang (Zheng et al., 2024) stores requests in a radix tree and uses longest prefix matching to increase cache reuse across requests. Preble (Srivatsa et al., 2025) jointly optimizes prefix reuse and load balancing across multiple GPUs, while BatchLLM (Zheng et al., 2026) groups requests with common prefixes offline and coordinates their execution. Dexter et al. (2026) study optimal request scheduling under a certain arrival pattern when cached requests are stored in a radix tree, and they propose a new family of scheduling policies which interpolate between first-come-first-served and longest prefix matching. Another related line of work studies fairness in LLM serving (Sheng et al., 2024; Cao et al., 2025). Our inference auction is the first approach that prioritizes economic efficiency.

Priority Auctions and Queues. The idea of allowing delay-sensitive users to purchase queue priority has a long history (e.g., Kleinrock, 1967; Glazer & Hassin, 1986; Hassin, 1995; Mitra, 2001; Afeche & Mendelson, 2004; Kittsteiner & Moldovanu, 2005; Heydenreich et al., 2008). Our setting differs from these classical models because inference requests are not computationally independent from one another. To the best of our knowledge, our work is the first to design incentive-compatible priority mechanisms for LLM inference serving.

Autobidding Agents. Autobidding has been studied extensively in online ad auctions; see Aggarwal et al. (2024) for a recent survey. Balseiro & Gur (2019) provide an adaptive pacing algorithm which scales bids according to realized expenditure. Balseiro et al. (2020) generalize their algorithm to online mirror descent and weaken several theoretical assumptions. Subsequent work studies social welfare and learning dynamics when autobidders interact in repeated ad auctions (e.g. Balseiro et al., 2021; Deng et al., 2021; Gaitonde et al., 2022; Lucier et al., 2023; Paes Leme et al., 2024; Anagnostides et al., 2026b). While our pacing autobidding algorithm is similar to some that are typically used in ad auctions, our welfare guarantees leverage the specific structure of the radix tree and are unique to our LLM inference setting.

LLMs and Incentives. Our results are broadly related to the growing body of work at the intersection of game theory, mechanism design, and generative AI. Immorlica et al. (2024) consider AI models as economic agents, and Rothschild et al. (2026) explore the implications of an agentic economy. Chen et al. (2026) and Liu et al. (2026) study mechanism design for LLM routing, while Harris et al. (2026) and Anagnostides et al. (2026a) design incentive-aware credit assignment schemes for AI-generated content. There is also a line of work which designs auctions for ad placements in LLM-generated text (Duetting et al., 2024; Dubey et al., 2024; Hajiaghayi et al., 2024).

7 Conclusion

We design the first *inference auction*, a mechanism for allocating LLM serving capacity during times of high request volume. Our inference auction enjoys the following key properties: (1) forcing the winning schedule to be a DFS schedule ensures cache optimality, (2) welfare maximization corresponds to a simple DFS comparator that orders radix subtrees by average bid, and (3) VCG payments, which incentivize truthful bidding, can be implemented with negligible computational overhead. We also design a pacing autobidder that learns how to bid in repeated inference auctions under a budget constraint. Our autobidder dynamically adjusts bids based on realized spending, and achieves sublinear regret relative to the optimal-in-hindsight bidding policy. When users in a batch run pacing autobidders independently, we bound the resulting welfare loss by the differences in the effective pacing factors of sibling subtrees. In experiments, our inference auction improves social welfare while retaining the caching and latency advantages of SGLang. Our work demonstrates how economic objectives can be incorporated directly into LLM serving infrastructure without sacrificing performance.

There are several exciting directions for future research. Serving large-scale systems may require joint orchestration of auctions, request routing, and cache management across multiple inference workers. Support for richer latency preferences is also an interesting direction. Finally, collusion-proof inference auction design is important to prevent strategic coordination across users.

AI USE STATEMENT

In this work, we used generative AI tools for finding related work, polishing the writing, some code generation for experiments, and aiding in proving some mathematical claims. We have not used generative AI tools for drafting sections of the paper, developing theoretical models or conceptual frameworks, interpreting results, generating a synthetic dataset, or for anything else. We have reviewed all AI-assisted work. We manually checked that all LLM-suggested references were relevant and we did our own manual literature survey as well, LLM-generated code was verified and tested for correctness, and LLM-generated proofs were manually rewritten and checked. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Philipp Afeche and Haim Mendelson. Pricing and priority auctions in queueing systems with a generalized delay cost structure. *Management Science*, 50(7):869–882, 2004.

Gagan Aggarwal, Ashwinkumar Badanidiyuru, Santiago R Balseiro, Kshipra Bhawalkar, Yuan Deng, Zhe Feng, Gagan Goel, Christopher Liaw, Haihao Lu, Mohammad Mahdian, et al. Auto-bidding and auctions in online advertising: A survey. *ACM SIGecom Exchanges*, 22(1):159–183, 2024.

Amazon Web Services. Optimize costs with amazon bedrock provisioned throughput and service tiers, 2026. URL `https://docs.aws.amazon.com/bedrock/latest/userguide/service-tiers-inference.html`.

Ioannis Anagnostides, Kshipra Bhawalkar, Christopher Liaw, Aranyak Mehta, Renato Paes Leme, Yifeng Teng, Grigoris Velegkas, and Weiqiang Zheng. Compensation design. *arXiv preprint arXiv:2607.14438*, 2026a.

Ioannis Anagnostides, Ian Gemp, Georgios Piliouras, and Kelly Spendlove. Chaos in autobidding auctions. *arXiv preprint arXiv:2602.09118*, 2026b.

Anthropic. Service tiers, 2026. URL `https://platform.claude.com/docs/en/api/service-tiers`.

Maria-Florina Balcan, Siddharth Prasad, and Tuomas Sandholm. Bicriteria multidimensional mechanism design with side information. *Advances in Neural Information Processing Systems*, 36: 40832–40852, 2023.

Santiago Balseiro, Haihao Lu, and Vahab Mirrokni. Dual mirror descent for online allocation problems. In *International Conference on Machine Learning*, pp. 613–628. PMLR, 2020.

Santiago Balseiro, Yuan Deng, Jieming Mao, Vahab Mirrokni, and Song Zuo. Robust auction design in the auto-bidding world. *Advances in Neural Information Processing Systems*, 34:17777–17788, 2021.

Santiago R Balseiro and Yonatan Gur. Learning in repeated auctions with budgets: Regret minimization and equilibrium. *Management Science*, 65(9):3952–3968, 2019.

Shiyi Cao, Yichuan Wang, Ziming Mao, Pin-Lun Hsu, Liangsheng Yin, Tian Xia, Dacheng Li, Shu Liu, Yineng Zhang, Yang Zhou, et al. Locality-aware fair scheduling in LLM serving. *arXiv preprint arXiv:2501.14312*, 2025.

Haolong Chen, Zhengyuan Xin, Liang Zhang, Lei Xue, and Guangxu Zhu. Error-aware reverse auction mechanism for large language model routing. *arXiv preprint arXiv:2608.12719*, 2026.

Edward H Clarke. Multipart pricing of public goods. *Public Choice*, pp. 17–33, 1971.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Yuan Deng, Jieming Mao, Vahab Mirrokni, and Song Zuo. Towards efficient auctions in an auto-bidding world. In *Proceedings of the Web Conference 2021*, pp. 3965–3973, 2021.

Gregory Dexter, Shao Tang, Ata Fatahi, Qingquan Song, Tejas Dharamsi, and Aman Gupta. LLM query scheduling with prefix reuse and latency constraints. *Advances in Neural Information Processing Systems*, 38:94356–94379, 2026.

Avinava Dubey, Zhe Feng, Rahul Kidambi, Aranyak Mehta, and Di Wang. Auctions with LLM summaries. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 713–722, 2024.

Paul Duetting, Vahab Mirrokni, Renato Paes Leme, Haifeng Xu, and Song Zuo. Mechanism design for large language models. In *Proceedings of the ACM Web Conference 2024*, pp. 144–155, 2024.

Jason Gaitonde, Yingkai Li, Bar Light, Brendan Lucier, and Aleksandrs Slivkins. Budget pacing in repeated auctions: Regret and efficiency without convergence. *arXiv preprint arXiv:2205.08674*, 2022.

Amihai Glazer and Refael Hassin. Stable priority purchasing in queues. *Operations Research Letters*, 4(6):285–288, 1986.

Theodore Groves. Incentives in teams. *Econometrica*, pp. 617–631, 1973.

MohammadTaghi Hajiaghayi, Sébastien Lahaie, Keivan Rezaei, and Suho Shin. Ad auctions for LLMs via retrieval augmented generation. *Advances in Neural Information Processing Systems*, 37:18445–18480, 2024.

Keegan Harris, Siddharth Prasad, and Asher Trockman. In-context credit assignment via the core. *arXiv preprint arXiv:2605.06920*, 2026.

Rafael Hassin. Decentralized regulation of a queue. *Management Science*, 41(1):163–173, 1995.

Birgit Heydenreich, Debasis Mishra, Rudolf Müller, and Marc Uetz. Optimal mechanisms for single machine scheduling. In *International Workshop on Internet and Network Economics*, pp. 414–425. Springer, 2008.

Nicole Immorlica, Brendan Lucier, and Aleksandrs Slivkins. Generative AI as economic agents. *ACM SIGecom Exchanges*, 22(1):93–109, 2024.

Thomas Kittsteiner and Benny Moldovanu. Priority auctions and queue disciplines that depend on processing time. *Management Science*, 51(2):236–248, 2005.

Leonard Kleinrock. Optimum bribing for queue position. *Operations Research*, 15(2):304–318, 1967.

Vijay Krishna and Motty Perry. Efficient mechanism design. *Available at SSRN 64934*, 1998.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Hongze Liu, Chang Guo, Yingzeng Li, Mengru Wang, Jiong Lou, Shijing Yuan, Hefeng Zhou, Chentao Wu, and Jie Li. Iemas: An incentive-efficiency routing framework for open agentic web ecosystems. *arXiv preprint arXiv:2603.17302*, 2026.

Brendan Lucier, Sarath Pattathil, Aleksandrs Slivkins, and Mengxiao Zhang. Autobidders with budget and ROI constraints: Efficiency, regret, and pacing dynamics. *arXiv preprint arXiv:2301.13306*, 2023.

Manipushpak Mitra. Mechanism design in queueing problems. *Economic Theory*, 17(2):277–305, 2001.

OpenAI. Priority processing, 2026. URL https://developers.openai.com/api/docs/guides/priority-processing.

Renato Paes Leme, Georgios Piliouras, Jon Schneider, Kelly Spendlove, and Song Zuo. Complex dynamics in autobidding systems. In *Proceedings of the 25th ACM Conference on Economics and Computation*, pp. 75–100, 2024.

Siddharth Prasad, Maria Florina Balcan, and Tuomas Sandholm. Weakest bidder types and new core-selecting combinatorial auctions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 17206–17214, 2026.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.

David M Rothschild, Markus M Mobius, Jake M Hofman, Eleanor Dillon, Daniel G Goldstein, Nicole Immorlica, Sonia Jaffe, Brendan Lucier, Aleksandrs Slivkins, and Matthew Vogel. The agentic economy. *Communications of the ACM*, 69(2):39–42, 2026.

Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 965–988, 2024.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for LLM serving. In *International Conference on Learning Representations*, volume 2025, pp. 37057–37082, 2025.

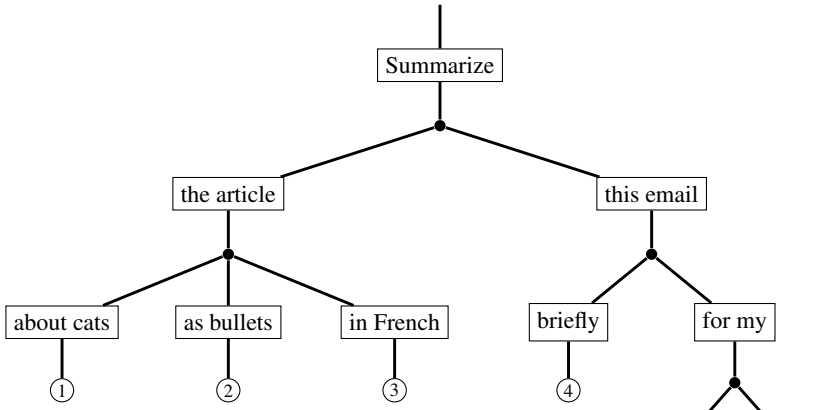
William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. SGLang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

Zhen Zheng, Xin Ji, Taosong Fang, Fanghao Zhou, Chuanjie Liu, and Gang Peng. BatchLLM: Optimizing large batched LLM inference with global prefix sharing and throughput-oriented token batching. *Proceedings of Machine Learning and Systems*, 8:1742–1756, 2026.



1 Summarize the article about cats
2 Summarize the article as bullets
3 Summarize the article in French
4 Summarize this email briefly
5 Summarize this email for my boss
6 Summarize this email for my team

Figure 5: A request radix tree. Each request is a root-to-leaf path; requests with a common prefix share the path from the root to their lowest common ancestor, and non-branching paths are contracted into a single edge. The maximum branching factor of this tree is 3. Requests in the same subtree are easier to store together in the KV cache, as their shared prefixes only need to be stored once.

A Appendix for Section 2: Cache-Optimal Schedules and Our Bidding Environment

Proposition 2.4. *For any* $\mathbf{v}_{1:m}$, $\max_{\sigma\in\mathcal{D}(\mathsf{T})} W(\sigma;\mathbf{v}_{1:m}) \geq \frac{1}{2}\max_{\sigma\in\mathcal{S}_n} W(\sigma;\mathbf{v}_{1:m})$.

Proof. Let $\sigma \in \mathcal{D}(\mathsf{T})$ be any feasible schedule and let π be the reverse of σ. We have $\pi \in \mathcal{D}(\mathsf{T})$ since π is attainable by reversing all child orderings in the DFS traversal of T corresponding to σ. For any user i and request r, we have $\mathbf{x}_i(\sigma)[r] + \mathbf{x}_i(\pi)[r] = 1$, so $W(\sigma;\mathbf{v}_{1:m}) + W(\pi;\mathbf{v}_{1:m}) = \sum_{i=1}^m \|\mathbf{v}_i\|_1$. Therefore, at least one of σ or π generates welfare at least $\frac{1}{2}\sum_{i=1}^m \|\mathbf{v}_i\|_1$, and so $\max_{\sigma\in\mathcal{D}(\mathsf{T})} W(\sigma;\mathbf{v}_{1:m}) \geq \frac{1}{2}\sum_{i=1}^m \|\mathbf{v}_i\|_1 \geq \frac{1}{2}\max_{\sigma\in\mathcal{S}_n} W(\sigma;\mathbf{v}_{1:m})$. □

B Appendix for Section 3: Welfare-Optimal Inference Auctions

Discussion on VCG Payments. The Clarke (1971) pivot term $h_i(\mathbf{b}_{-i})$ in VCG payments is often (informally) framed as the maximum welfare the bidders excluding i could obtain when bidder i is removed from the auction, while we define it by zeroing out that bidder's bids. In typical multi-item multi-bidder auction settings, there is no distinction. In our setting, actually evicting i's requests would alter the radix tree of requests and change the ground set of feasible schedules to allow requests to go un-served. In addition to the fact that all requests should eventually be served, a payment scheme based on eviction would incur outward payments made by the system to the users. Indeed, any zero-bidder would have to be paid because the maximum welfare enjoyed by the remaining bidders decreases when that zero-bidder is evicted, since each remaining bidder would win lower fractional priority.

In our setting, the correct formulation of VCG involves the minimum value of the maximum welfare the bidders excluding i could obtain over all possible "types" (bids) for bidder i, which is attained when i is a zero bidder. This is precisely the reasoning folded into the enhancement of VCG based on "weakest types" (see, e.g., Krishna & Perry, 1998; Balcan et al., 2023; Prasad et al., 2026).

For completeness, we include below the standard arguments demonstrating that VCG in our setting is indeed efficient, incentive compatible, individually rational, and incurs no positive transfers.

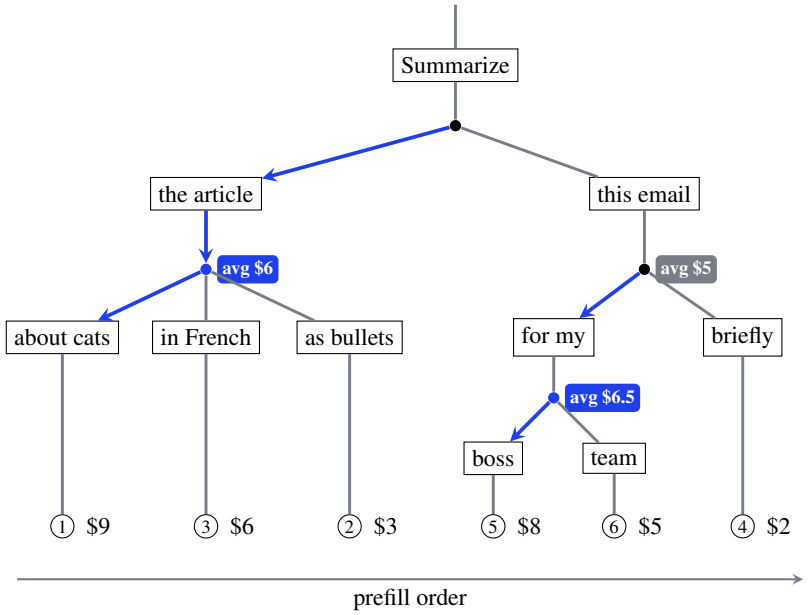


Figure 6: The welfare-maximizing DFS schedule on an example radix tree. At every internal node, children are visited in decreasing order of average subtree bid (blue marks the child visited first). The resulting schedule is $1, 3, 2, 5, 6, 4$. Note that request 5 bids \$8 but is served after request 2, which bids \$3. This is because request 5 belongs to a subtree with a lower average bid, and a DFS schedule cannot interleave the two subtrees.

Under VCG prices, efficiency follows from Theorem 3.1. To see why these payments incentivize truthful bidding, fix the bids $\mathbf{b}_{-i}$ of the other users. If user i bids according to $\mathbf{b}_i$, his utility can be written as

$$V_i(\bar{A}(\mathbf{b}_i, \mathbf{b}_{-i})) - p_i^{\text{VCG}}(\mathbf{b}_i, \mathbf{b}_{-i}) = W(\bar{A}(\mathbf{b}_i, \mathbf{b}_{-i}); (\mathbf{v}_i, \mathbf{b}_{-i})) - h_i(\mathbf{b}_{-i}).$$

$h_i(\mathbf{b}_{-i})$ does not depend on $\mathbf{b}_i$, and $W(\bar{A}(\mathbf{b}_i, \mathbf{b}_{-i}); (\mathbf{v}_i, \mathbf{b}_{-i}))$ is maximized when $\mathbf{b}_i = \mathbf{v}_i$ by Theorem 3.1. Therefore, it is a (weakly) dominant strategy for each user to bid truthfully, independent of the bids of the other users. Individual rationality holds because user i's values and priorities are non-negative. Under truthful bidding, user i's utility is $\max_{\sigma \in \mathcal{D}(\mathsf{T})} W(\sigma; (\mathbf{v}_i, \mathbf{b}_{-i})) - h_i(\mathbf{b}_{-i})$. Now consider the schedule that maximizes the other users' welfare when i's bids are zero. That schedule remains feasible when i bids truthfully. Under it, total welfare is $h_i(\mathbf{b}_{-i}) + V_i(\sigma) \geq h_i(\mathbf{b}_{-i})$, since $V_i(\sigma) \geq 0$. Finally, no positive transfers follows because $h_i(\mathbf{b}_{-i})$ is the maximum welfare available to the other users, and is always at least the welfare they receive under the selected schedule.

B.1 Algorithm for Computing the Welfare-Maximizing Schedule

Theorem 3.1. *Given bid vectors $\mathbf{b}_{1:m}$, the average-bid DFS scheduling rule $\bar{A}(\mathbf{b}_{1:m})$ given by Algorithm 1 returns a cache-optimal schedule σ that maximizes $W(\sigma; \mathbf{b}_{1:m})$ among all DFS schedules. Furthermore, this schedule can be computed in $O(M + n \log \nu)$ expected run-time.*

Let $\Phi(\sigma; \mathbf{b}_{1:m}) \coloneqq (n-1)W(\sigma; \mathbf{b}_{1:m}) = \sum_{r=1}^{n} b_r \cdot |\{j : \sigma(r) < \sigma(j)\}| = \sum_{r=1}^{n} \sum_{j \succ_\sigma r} b_r$. Maximizing $W(\cdot; \mathbf{b}_{1:m})$ over a set of feasible schedules is equivalent to maximizing its un-normalized version $\Phi(\cdot; \mathbf{b}_{1:m})$.

Proof. Let $\sigma \in \mathcal{D}(\mathsf{T})$ be a valid DFS leaf ordering. For u an internal node (that is, not a leaf node) of T, let $\mathsf{c}_1, \ldots, \mathsf{c}_{d(\mathsf{u})}$ be the children of u, labeled in the DFS order induced by σ. So, all leaves under child c_j are visited before all leaves under child c_k for all $1 \leq j < k \leq d(\mathsf{u})$. The un-normalized welfare contribution from ordered leaf pairs in $\mathcal{L}(\mathsf{c}_j) \times \mathcal{L}(\mathsf{c}_k)$ for $j < k$ is therefore $\sum_{i_1 \in \mathcal{L}(\mathsf{c}_j)} \sum_{i_2 \in \mathcal{L}(\mathsf{c}_k)} b_{i_1} = N(\mathsf{c}_k) F_b(\mathsf{c}_j)$. It follows that un-normalized welfare can be written as

$$\Phi(\sigma; \mathbf{b}_{1:m}) = \sum_{\mathsf{u}} \sum_{1 \leq j < k \leq d(\mathsf{u})} N(\mathsf{c}_k) F_b(\mathsf{c}_j).$$

We show that $\Phi(\sigma; \mathbf{b}_{1:m})$ is maximized by visiting the children of any internal node u in descending order of F_b/N. Suppose $\mathsf{c}_j, \mathsf{c}_{j+1} \in \mathcal{C}(\mathsf{u})$ are adjacent in the DFS traversal with $F_b(\mathsf{c}_j)/N(\mathsf{c}_j) < F_b(\mathsf{c}_{j+1})/N(\mathsf{c}_{j+1})$. This pair contributes $N(\mathsf{c}_{j+1})F_b(\mathsf{c}_j)$ to Φ. Swapping their order contributes $N(\mathsf{c}_j)F_b(\mathsf{c}_{j+1}) > N(\mathsf{c}_{j+1})F_b(\mathsf{c}_j)$ to welfare without impacting the contribution of any other pair in $\mathcal{C}(\mathsf{u})$ (since the relative ordering of c_j and c_{j+1} with respect to all other children is preserved). Therefore, each inner sum of $\sum_{1 \le j < k \le d(\mathsf{u})} N(\mathsf{c}_k)F_b(\mathsf{c}_j)$ is independently maximized for each u by visiting $\mathcal{C}(\mathsf{u})$ in descending order of average bid, so average-bid DFS is a welfare optimal DFS traversal.

Runtime analysis: A radix tree can be constructed in $O(M)$ expected time using hash tables for child lookup. Let $\mathsf{T} = (V, E)$ be the radix tree and let $d(\mathsf{u})$ denote the number of children of internal node u. Note that $d(\mathsf{u}) \le \nu$ by definition, and let I be the total number of internal nodes. It takes $O(|V|)$ time to compute $F_b(\mathsf{u})$ for every node, since each node is processed a constant number of times across all computations. Sorting at each internal node u takes $O(d(\mathsf{u}) \log(d(\mathsf{u})))$ time. The time it takes to sort all nodes is $O(\sum_{\mathsf{u}} d(\mathsf{u}) \log(d(\mathsf{u}))) = O(\sum_{\mathsf{u}} d(\mathsf{u}) \log \nu) = O(|E| \log \nu) = O(|V| \log \nu)$. A DFS schedule following this order outputs the order in $O(|V|)$ time. Finally, since a radix tree has no internal node with exactly one child, we have that $n + I - 1 = |E| = \sum_u d(u) \ge 2I$, and so $V \le 2n - 1$. □

B.2 Algorithm for Computing VCG Prices

In this subsection we abbreviate F_b as F for brevity. Let $[x]_+ \coloneqq \max\{x, 0\}$.

Fix an internal node u and let $\mathsf{c}_1, \ldots, \mathsf{c}_d$ denote its children sorted in descending average-bid order, so

$$\frac{F(\mathsf{c}_1)}{N(\mathsf{c}_1)} \ge \cdots \ge \frac{F(\mathsf{c}_d)}{N(\mathsf{c}_d)}.$$

Consider some user i. Let

$$Z_{\mathsf{c}_j}[i] \coloneqq \sum_{r \in \mathcal{R}_i \cap \mathcal{L}(\mathsf{c}_j)} b_r$$

denote user i's aggregate bid across his requests under child c_j. After zeroing out user i's bids, the total bid under child c_j is

$$G_i(\mathsf{c}_j) \coloneqq F(\mathsf{c}_j) - Z_{\mathsf{c}_j}[i].$$

Lemma B.1. *For any user i and internal node u, the welfare externality at u due to i is*

$$E_i(\mathsf{u}) \coloneqq \sum_{1 \le j < k \le d} \left[G_i(\mathsf{c}_k)N(\mathsf{c}_j) - G_i(\mathsf{c}_j)N(\mathsf{c}_k) \right]_+ .$$

Consequently, $(n-1)p_i^{\mathtt{VCG}} = \sum_{\mathsf{u}} E_i(\mathsf{u})$.

Proof. In the original descending average-bid order, pair $j < k$ of children contributes $G_i(\mathsf{c}_j)N(\mathsf{c}_k)$ to the other users' welfare. The counterfactual optimal ordering sorts children in decreasing order of G_i/N, so $j < k$ contributes $\max\{G_i(\mathsf{c}_j)N(\mathsf{c}_k), G_i(\mathsf{c}_k)N(\mathsf{c}_j)\}$ to welfare. So, the externality—that is, the welfare gain in the counterfactual optimum—is precisely $[G_i(\mathsf{c}_k)N(\mathsf{c}_j) - G_i(\mathsf{c}_j)N(\mathsf{c}_k)]_+$. Summing over all child pairs yields the claimed externality $E_i(\mathsf{u})$. To compute the VCG price it suffices to sum $E_i(\mathsf{u})$ over all internal nodes and normalize. □

Preprocessing. For $t = 1, \ldots, d$, let

$$P_F[t] \coloneqq \sum_{k=1}^{t} F(\mathsf{c}_k); \quad P_N[t] \coloneqq \sum_{k=1}^{t} N(\mathsf{c}_k).$$

Computing these prefix sums is a $O(d(\mathsf{u}))$ run-time operation at each node. So, the overall run-time of prefix sum computation is $O(\sum_{\mathsf{u}} d(\mathsf{u})) = O(n)$. This computation is done once and the same prefix sums are used in the price calculation for all bidders.

B.2.1 COMPUTING LOCAL EXTERNALITIES

Let $\mathcal{A}_{\mathsf{u}}[i] \coloneqq \{j \in \{1,\dots,d\} : Z_{\mathsf{c}_j}[i] > 0\}$ denote the set of affected child indices; enumerate $\mathcal{A}_{\mathsf{u}}[i]$ as $j_1 < \dots < j_a$ (so $F(\mathsf{c}_{j_1})/N(\mathsf{c}_{j_1}) \geq \dots \geq F(\mathsf{c}_{j_a})/N(\mathsf{c}_{j_a})$). We now show how to compute $E_i(\mathsf{u})$ at a given node, assuming that at each internal node we have access to F, N, P_F, N_F, $\mathcal{A}_{\mathsf{u}}[i]$, and $Z_{\mathsf{c}_{j_s}}[i]$ for all $j_s \in \mathcal{A}_{\mathsf{u}}[i]$. (We will show how to perform bottom-up computation of $\mathcal{A}$ and Z in Section B.2.2.)

Pairs of children are of three types: unaffected-unaffected, affected-unaffected, affected-affected. Unaffected-unaffected pairs have zero externality since zeroing out i's bids does not change their relative ordering.

Affected-unaffected pairs. Let c_{j_s} be an affected child whose average bid has decreased from $F(\mathsf{c}_{j_s})/N(\mathsf{c}_{j_s})$ to $G_i(\mathsf{c}_{j_s})/N(\mathsf{c}_{j_s})$. An unaffected child before c_{j_s}, that is, an unaffected child c_k with $k < j_s$, stays before c_{j_s}. An unaffected child c_k with $k > j_s$ overtakes c_{j_s} if and only if $F(\mathsf{c}_k)/N(\mathsf{c}_k) > G_i(\mathsf{c}_{j_s})/N(\mathsf{c}_{j_s})$.

Run binary search to find

$$p_s \coloneqq \max\left\{k \in \{j_s,\dots,d\} : \frac{F(\mathsf{c}_k)}{N(\mathsf{c}_k)} > \frac{G_i(\mathsf{c}_{j_s})}{N(\mathsf{c}_{j_s})}\right\}.$$

So, all unaffected children in the interval $(j_s, p_s]$ overtake c_{j_s} in the counterfactual optimal ordering. Let $U_s \coloneqq (j_s, p_s] \cap ([d] \setminus \mathcal{A}_i(\mathsf{u}))$ denote this set of unaffected children. The externality due to all such unaffected children $\mathsf{c}_k, k \in U_s$, from overtaking c_{j_s} is $F(\mathsf{c}_k)N(\mathsf{c}_{j_s}) - G_i(\mathsf{c}_{j_s})N(\mathsf{c}_k)$, and so the total externality from all affected-unaffected child pairs is

$$\sum_{s=1}^{a}\left(N(\mathsf{c}_{j_s})\sum_{k\in U_s}F(\mathsf{c}_k) - G_i(\mathsf{c}_{j_s})\sum_{k\in U_s}N(\mathsf{c}_k)\right).$$

We now show how to compute this quantity via prefix sums. For $s = 1,\dots,a$ compute affected child prefix sums

$$P_F^A[s] \coloneqq \sum_{\ell=1}^{s}F(\mathsf{c}_{j_\ell}); \quad P_N^A[s] \coloneqq \sum_{\ell=1}^{s}N(\mathsf{c}_{j_\ell}).$$

Run binary search to compute $q_s \coloneqq \max\{\ell : j_\ell \leq p_s\}$. Then,

$$\sum_{k\in U_s}F(\mathsf{c}_k) = \underbrace{(P_F[p_s] - P_F[j_s])}_{\substack{\text{total bid under}\\ \text{children in } (j_s,p_s]}} - \underbrace{\left(P_F^A[r_s] - P_F^A[s]\right)}_{\substack{\text{total bid under}\\ \text{affected children in } (j_s,p_s]}} =: S_F[s]$$

and

$$\sum_{k\in U_s}N(\mathsf{c}_k) = \underbrace{(P_N[p_s] - P_N[j_s])}_{\substack{\text{total leaves under}\\ \text{children in } (j_s,p_s]}} - \underbrace{\left(P_N^A[r_s] - P_N^A[s]\right)}_{\substack{\text{total leaves under}\\ \text{affected children in } (j_s,p_s]}} =: S_N[s].$$

So, the total externality from affected-unaffected pairs is

$$\sum_{s=1}^{a}\left(N(\mathsf{c}_{j_s})S_F[s] - G_i(\mathsf{c}_{j_s})S_N[s]\right). \tag{1}$$

Affected-affected pairs. Sort the affected children in descending order of modified average bid. Let π denote the sorting order, so

$$\frac{G_i(\mathsf{c}_{j_{\pi(1)}})}{N(\mathsf{c}_{j_{\pi(1)}})} \geq \dots \geq \frac{G_i(\mathsf{c}_{j_{\pi(a)}})}{N(\mathsf{c}_{j_{\pi(a)}})}.$$

The total externality from pairs of affected children is

$$\sum_{1\leq s<t\leq a}G_i(\mathsf{c}_{j_{\pi(s)}})N(\mathsf{c}_{j_{\pi(t)}}) - \sum_{1\leq s<t\leq a}G_i(\mathsf{c}_{j_s})N(\mathsf{c}_{j_t}). \tag{2}$$

ALGORITHM 4: VCG Payments

Initialize $P_i \leftarrow 0$ for each user i
Let u be the root of the radix tree T
PROCESS(u)
return $[P_i/(n-1)]_{i=1}^m$

ALGORITHM 5: PROCESS(u)

if u is a leaf with request $r \in \mathcal{R}_i$ **then**
  if $b_r > 0$ **then**
    $Z_{\mathsf{u}} \leftarrow \{i \mapsto b_r\}$
  else if $b_r = 0$ **then**
    $Z_{\mathsf{u}} \leftarrow \emptyset$
  return Z_{u}
Initialize $Z_{\mathsf{u}} \leftarrow \emptyset, \mathcal{A}_{\mathsf{u}} \leftarrow \emptyset$
for $j = 1, \ldots, d(\mathsf{u})$ **do**
  $Z_{\mathsf{c}_j} \leftarrow$ PROCESS(c_j)
  for $(i, z) \in Z_{\mathsf{c}_j}$ **do**
    $\mathcal{A}_{\mathsf{u}}[i]$.append$((j, z))$
    $Z_{\mathsf{u}}[i] \leftarrow Z_{\mathsf{u}}[i] + z$
for $i \in \mathcal{A}_{\mathsf{u}}$.keys **do**
  $P_i \leftarrow P_i +$ LOCALEXTERNALITY$(\mathsf{u}, \mathcal{A}_{\mathsf{u}}[i])$
return Z_{u}

For any ordering $\ell_1, \ldots, \ell_a$ of affected children, a sum of the form $\sum_{1 \le s < t \le a} G_i(\mathsf{c}_{\ell_s}) N(\mathsf{c}_{\ell_t})$ can be computed in $O(a)$ run-time: initialize $(R, C) \leftarrow (\sum_{t=1}^{a} N(\mathsf{c}_{\ell_t}), 0)$; for $s = 1, \ldots, a$ do $R \leftarrow R - N(\mathsf{c}_{\ell_s}), C \leftarrow C + G_i(\mathsf{c}_{\ell_s}) R$; return C.

Finally, the total externality $E_i(\mathsf{u})$ is the sum of expressions (1) and (2). We perform for each affected child one binary search on a list of size $O(d(\mathsf{u}))$ and another on a list of size a. Prefix sum computation of P_F^A, P_N^A and affected-affected pair externality computation are $O(a)$ run-time computations. Sorting takes run-time $O(a \log a)$. So, the overall run-time for computing $E_i(\mathsf{u})$ is $O(a \log d(\mathsf{u}))$.

B.2.2 COMPUTING ALL PRICES

Let LOCALEXTERNALITY$(\mathsf{u}, \mathcal{A}_u[i])$ return $E_i(\mathsf{u})$, where $\mathcal{A}_u[i] = [(j_1, Z_{\mathsf{c}_{j_1}}[i]), \ldots, (j_a, Z_{\mathsf{c}_{j_a}}[i])]$ is a list containing the children of u affected by i and $Z_{\mathsf{c}_{j_s}}[i]$ is i's aggregate bid under c_{j_s}.

Algorithm 4 computes VCG payments via a bottom-up computation of affected children and aggregate bids (Algorithm 5).

B.2.3 RUN-TIME ANALYSIS

Global prefix sum arrays P_F, P_N can be constructed in $O(\sum_{\mathsf{u}} d(\mathsf{u})) = O(n)$ run-time.

Let $\Gamma := \sum_{\mathsf{u}} \sum_i |A_{\mathsf{u}}[i]|$. Observe that $\Gamma = \sum_{\mathsf{u}} \sum_{j=1}^{d(\mathsf{u})} |Z_{\mathsf{c}_j}| = \sum_{\mathsf{u} \neq \mathrm{root}(\mathsf{T})} |Z_{\mathsf{u}}|$. Indeed, every entry $(i, z) \in Z_{\mathsf{c}_j}$ appends exactly one pair (j, z) to $\mathcal{A}_{\mathsf{u}}[i]$, and every element of every affected-child list arises in this way.

PROCESS called on the root of T visits every node once, and every entry of the hash map Z_{c_j} is visited. Each call to LOCALEXTERNALITY$(u, \mathcal{A}_{\mathsf{u}}[i])$ incurs $O(|\mathcal{A}_{\mathsf{u}}[i]| \log d(\mathsf{u}))$ run-time. So, the total run-time across all calls to LOCALEXTERNALITY is $O(\sum_{\mathsf{u}} \sum_i |\mathcal{A}_{\mathsf{u}}[i]| \log d(\mathsf{u})) \le O(\Gamma \log \nu)$.

The total run-time of PROCESS on the root of T (Algorithm 4) is therefore $O(n + \Gamma \log \nu)$.

We next bound Γ in terms of the input size. For each non-root node u and each entry $(i, z) \in Z_{\mathsf{u}}$, charge this entry to an arbitrary positive-bid request $r \in \mathcal{R}_i \cap \mathcal{L}(\mathsf{u})$. A request r can receive at most one charge from each non-root node on its root-to-leaf path, and so its total charge is at most $\mathrm{depth}_{\mathsf{T}}(r)$. Consequently, $\Gamma = \sum_{\mathsf{u} \neq \mathrm{root}(\mathsf{T})} |Z_{\mathsf{u}}| \le \sum_{r=1}^{n} \mathrm{depth}_{\mathsf{T}}(r) \le M$.

Also, $\Gamma = O(mn)$, since each map Z_{u} contains at most one entry per user and the radix tree has $O(n)$ nodes. So, $\Gamma = O(\min\{M, mn\})$.

We have therefore proven Theorem 3.3.

Theorem 3.3. *Given as input the processed radix tree T, where at every node u (1) $F_b(\mathsf{u})$ and $N(\mathsf{u})$ are stored and (2) its children are sorted in decreasing order of average bid, all VCG payments $p_1^{\mathtt{VCG}}(\mathbf{b}_{1:m}), \ldots, p_m^{\mathtt{VCG}}(\mathbf{b}_{1:m})$ can be computed in $O(\min\{M, mn\} \log \nu)$ total run-time.*

C Appendix for Section 4: Autobidding Agents for Inference Auctions

Theorem 4.2. *Suppose that $(\mathbf{v}_1^{(t)}, \mathsf{T}^{(t)}, \mathbf{b}_{2:m}^{(t)})_{t=1}^T$ are drawn i.i.d. from an unknown, finite support distribution $\mathcal{P}$. Let $K := \max_{t\in[T]} |\mathcal{R}_1^{(t)}|$, $\|\mathbf{v}_1^{(t)}\|_\infty \le C$, and suppose that schedules and payments are assigned according to the inference auction of Section 3. Then if $\eta = \Theta(1/\sqrt{T})$, Algorithm 3 obtains regret $R(T) = O(\mathrm{poly}(C, K, \gamma)\sqrt{T})$ when T is sufficiently large.*

Proof Sketch. The framework of Balseiro et al. (2020) requires (1) a convex and bounded action set that contains a "do nothing" action, (2) nonnegative, bounded, concave rewards, and (3) nonnegative, bounded resource consumption. To obtain the required action set, we first subtract the priority received by a zero bid, which is independent of the autobidder's report, and consider only additional priority. For any fixed auction instance, the finitely many feasible schedules induce a finite menu of priority-and-payment outcomes. Allowing randomization over this menu produces a bounded and convex action set with zero as a feasible option, as well as nonnegative and bounded expected utility. Under VCG, the autobidder's paced bid selects exactly the menu option that maximizes incremental utility after penalizing its payment by the current budget multiplier. Thus, the two algorithms coincide until the remaining budget is no greater than the largest possible payment. Finally, the randomized hindsight benchmark is at least as strong as our original deterministic benchmark in Definition 4.1. □

	Our Notation	Balseiro et al. (2020)
Time Horizon:	T	T
Capacity:	$T\gamma$	$T\rho$
Action Set:	$\mathcal{Y} = \{\mathbf{y} \ge \mathbf{0} : \sum_k \mathbf{y}[k] \le 1\}$	Generic action set $\mathcal{X}$
Decision Variable:	Menu selection vector $y \in \mathcal{Y}$	Action $x_t \in \mathcal{X}$
Reward Function:	$\sum_k (\langle \mathbf{v}_1^{(t)}, \mathbf{a}^{(t)}[k]\rangle - \mathbf{p}^{(t)}[k])\mathbf{y}[k]$	$f_t(x)$
Resource Consumption:	$\langle \mathbf{y}, \mathbf{p}^{(t)}\rangle$	$b_t x_t$
Dual Variable:	$\mu^{(t)}$	μ_t
Primal Objective:	$\max_{\mathbf{y}\in\mathcal{Y}} \sum_k (\langle \mathbf{v}_1^{(t)}, \mathbf{a}^{(t)}[k]\rangle - (1+\mu^{(t)})\mathbf{p}^{(t)}[k])\mathbf{y}[k]$	$\max_{x_t\in\mathcal{X}}\{f_t(x_t) - \mu_t^\top b_t x_t\}$

Table 1: Notation translation between our setting and Balseiro et al. (2020).

Proof. Define the shorthand $\mathbf{x}^{(t)}(\mathbf{b}_1) := \mathbf{x}_1(\bar{A}(\mathbf{b}_1, \mathbf{b}_{2:m}^{(t)}))$. Because the zero bid vector can receive positive priority for free, define $\mathbf{x}^{(t,0)} := \mathbf{x}^{(t)}(\mathbf{0})$ and $\mathbf{a}^{(t)}(\mathbf{b}_1) := \mathbf{x}^{(t)}(\mathbf{b}_1) - \mathbf{x}^{(t,0)}$. Thus $\mathbf{a}^{(t)}(\mathbf{b}_1)$ is the vector of *incremental* priorities obtained through bidding according to vector $\mathbf{b}_1$ in round t. The baseline value $\sum_{t=1}^T \langle \mathbf{v}_1^{(t)}, \mathbf{x}^{(t,0)}\rangle$ is independent of the autobidder's reports and therefore does not affect either the optimal bidding policy or regret.

Fix a round t and hold the other users' requests and bids fixed. As the autobidder varies its bid vector $\mathbf{b}_1 \in \mathbb{R}_+^{|\mathcal{R}_1^{(t)}|}$, the VCG mechanism presents it with a finite menu of incremental-priority vector and payment pairs

$$\mathcal{M}^{(t)} := \left\{ \left(\mathbf{a}^{(t)}(\mathbf{b}_1), p^{(t)}(\mathbf{b}_1) \right) : \mathbf{b} \in \mathbb{R}_+^{|\mathcal{R}_1^{(t)}|} \right\}.$$

Because this menu is finite, we can vectorize its distinct elements as $\mathcal{M}^{(t)} = \{(\mathbf{a}^{(t)}[k], \mathbf{p}^{(t)}[k])\}_{k=1}^{K^{(t)}}$, where $\mathbf{a}^{(t)}[k]$ is itself a vector of length $|\mathcal{R}_1^{(t)}|$.[7] Since every menu outcome is induced by a feasible DFS schedule, we have that $K^{(t)} \le n!$. We can "convexify" the menu $\mathcal{M}^{(t)}$ by introducing a vector

[7] If $\langle \mathbf{v}_1^{(t)}, \mathbf{a}^{(t)}[k]\rangle - \mathbf{p}^{(t)}[k] < 0$, we replace the item with $(\mathbf{0}, 0)$. This is without loss of generality since the bidder can always obtain zero incremental utility by bidding nothing.

$\mathbf{y} \in \mathbb{R}_+^{n!}$ with $\sum_{k=1}^{n!} \mathbf{y}[k] \leq 1$, where we interpret $\mathbf{y}[k]$ as the probability of selecting menu item k.[8] Let $\mathcal{Y}$ be the set of all such vectors $\mathbf{y}$.

Define $f^{(t)}(\mathbf{y}) \coloneqq \sum_{k=1}^{n!}(\langle \mathbf{v}_1^{(t)}, \mathbf{a}^{(t)}[k]\rangle - \mathbf{p}^{(t)}[k])\mathbf{y}[k]$ and $\langle \mathbf{c}^{(t)}, \mathbf{y}\rangle \coloneqq \sum_{k=1}^{n!} \mathbf{p}^{(t)}[k]\mathbf{y}[k]$. This now is a valid instance of the online allocation framework of Balseiro et al. (2020) with one resource: The action set $\mathcal{Y}$ is convex, bounded, and contains the zero action. The reward function and resource consumption are both nonnegative and linear. Because the autobidder has at most K requests and every request value is at most C, $\langle \mathbf{v}_1^{(t)}, \mathbf{a}^{(t)}[k]\rangle \leq \|\mathbf{v}_1^{(t)}\|_1 \leq KC$. After replacing negative utility menu items with the zero item, every remaining menu item satisfies $0 \leq \langle \mathbf{v}_1^{(t)}, \mathbf{a}^{(t)}[k]\rangle - \mathbf{p}^{(t)}[k] \leq KC$ and $0 \leq \mathbf{p}^{(t)}[k] \leq \langle \mathbf{v}_1^{(t)}, \mathbf{a}^{(t)}[k]\rangle \leq KC$. Thus rewards and resource consumption are both bounded by KC, and the resource capacity is $T\gamma$.

Algorithm 3 is the same as Algorithm 1 in Balseiro et al. (2020) specialized to online gradient descent and using our notation, up to its stopping time. To see this, fix a round t and hold $\mathsf{T}^{(t)}$ and $\mathbf{b}_{2:m}^{(t)}$ fixed. Let $W_{-1}^{(t)}(\sigma)$ denote the welfare of all players except the autobidder under σ, and let $h_1^{(t)} \coloneqq \max_{\sigma \in \mathcal{D}(\mathsf{T}^{(t)})} W_{-1}^{(t)}(\sigma)$. Under VCG, bidding according to bid vector $\mathbf{b}_1$ selects

$$\bar{A}^{(t)}(\mathbf{b}_1, \mathbf{b}_{2:m}^{(t)}) \in \arg\max_{\sigma \in \mathcal{D}(\mathsf{T}^{(t)})} \{\langle \mathbf{b}_1, \mathbf{x}_1(\sigma)\rangle + W_{-1}^{(t)}(\sigma)\}$$

and incurs the user-level payment $p^{(t)}(\mathbf{b}_1, \mathbf{b}_{2:m}^{(t)}) = h_1^{(t)} - W_{-1}^{(t)}(\bar{A}^{(t)}(\mathbf{b}_1, \mathbf{b}_{2:m}^{(t)}))$. Substituting in the definitions of $\mathbf{a}^{(t)}$ and $p^{(t)}$ gives

$$\langle \mathbf{b}_1, \mathbf{a}^{(t)}(\mathbf{b}_1)\rangle - p^{(t)}(\mathbf{b}_1, \mathbf{b}_{2:m}^{(t)}) = \langle \mathbf{b}_1, \mathbf{x}^{(t)}(\mathbf{b}_1)\rangle + W_{-1}^{(t)}(\bar{A}^{(t)}(\mathbf{b}_1, \mathbf{b}_{2:m}^{(t)})) - (\langle \mathbf{b}_1, \mathbf{x}^{(t)}(\mathbf{0})\rangle + h_1^{(t)}).$$

For a fixed bid vector $\mathbf{b}_1$, the final term on the right-hand side does not depend on the schedule that is selected. Therefore, maximizing $\langle \mathbf{b}_1, \mathbf{a}^{(t)}(\mathbf{b}_1)\rangle - p^{(t)}(\mathbf{b}_1, \mathbf{b}_{2:m}^{(t)})$ is equivalent to maximizing $\langle \mathbf{b}_1, \mathbf{x}_1(\sigma)\rangle + W_{-1}^{(t)}(\sigma)$. This is exactly reported total welfare, which is precisely what the VCG allocation rule maximizes.

Using our notation, the primal step of the dual mirror descent algorithm of Balseiro et al. (2020) selects a menu item maximizing $\langle \mathbf{v}_1^{(t)}, \mathbf{a}\rangle - (1 + \mu^{(t)})p$. The VCG mechanism implements this menu choice when the submitted bid vector is $\tilde{\mathbf{b}}^{(t)} = \frac{\mathbf{v}_1^{(t)}}{1+\mu^{(t)}}$.

Let τ be the stopping time used by Balseiro et al. (2020), namely the first round after which cumulative spending is within CK of the total budget $T\gamma$. For every $t \leq \tau$, minimality of the stopping time implies that $B^{(t)} > CK$. Because $\|\alpha^{(t)}\mathbf{v}_1^{(t)}\|_1 \leq CK < B^{(t)}$ in every round $t \leq \tau$, both Algorithm 3 and Algorithm 1 in Balseiro et al. (2020) therefore execute the menu item induced by $\alpha^{(t)}\mathbf{v}_1^{(t)}$, incur the same payment, and perform the same dual update in every round through τ. The proof of Theorem 1 in Balseiro et al. (2020) bounds regret by the primal-dual gap accumulated through τ, together with a term $CK\mathbb{E}[T - \tau]$ accounting for the remaining rounds. The former depends only on the algorithms' behavior through τ, while their stopping-time bound controls the latter. Since rewards are nonnegative, the rewards earned after τ may be discarded when lower-bounding Algorithm 3's total reward. Thus the algorithms need not coincide after τ. All assumptions for Theorem 1 in Balseiro et al. (2020) now hold, and so translating it into our notation gives the desired regret guarantee. Finally, it is worth noting that the guarantees of Balseiro et al. (2020) are with respect to an adversary optimizing over $\mathcal{Y}$. However the convexified hindsight benchmark can only be larger than the deterministic hindsight optimum, so this bound holds in our setting as well.

The regret bound in Balseiro et al. (2020) give us the guarantee

$$R(T) \leq 2(C^2K^2 + \gamma^2)\eta T + \frac{CK}{\gamma\eta}\left(\frac{CK}{\gamma} + 1\right) + \frac{C^2K^2}{\gamma}.$$

Setting $\eta = \sqrt{\frac{\frac{CK}{\gamma}(\frac{CK}{\gamma}+1)}{2T(C^2K^2+\gamma^2)}}$ gives us a regret bound of

$$R(T) \leq 2\sqrt{\frac{CK}{\gamma}(\frac{CK}{\gamma} + 1) \cdot 2(C^2K^2 + \gamma^2)T} + \frac{C^2K^2}{\gamma},$$

[8] If $K^{(t)} < n!$, we can pad $\mathbf{y}$ with zeros at the end.

as long as $T \geq \frac{C^3K^3}{2\gamma(C^2K^2+\gamma^2)}(\frac{CK}{\gamma}+1)$.

□

Lemma C.1 (DFS Welfare Decomposition). *If a DFS schedule σ visits the children $\mathsf{c}_1, \ldots, \mathsf{c}_{d(\mathsf{u})}$ of each internal node u in that order, then*

$$(n-1)W(\sigma;\mathbf{v}) = \sum_{\mathsf{u}} \sum_{1\leq j<k\leq d(\mathsf{u})} N(\mathsf{c}_j)N(\mathsf{c}_k)\bar{v}(\mathsf{c}_j). \tag{3}$$

Consequently,

$$(n-1)\max_{\sigma\in\mathcal{D}(T)} W(\sigma;\mathbf{v}) = \sum_{\mathsf{u}} \sum_{\{\mathsf{c},\mathsf{d}\}\subseteq\mathcal{C}(\mathsf{u})} N(\mathsf{c})N(\mathsf{d})\max\{\bar{v}(\mathsf{c}),\bar{v}(\mathsf{d})\}. \tag{4}$$

Proof. Both equations follow from the same reasoning as in the proof of Theorem 3.1. □

Theorem 4.4. *Suppose that each request r has corresponding bid $b_r = \alpha_r v_r$ for pacing factors $\alpha_1, \ldots, \alpha_n \in (0,1]$, and let Ω_{u} denote the marginal contribution of node u to the optimal DFS welfare. Then the social welfare for the batch is bounded as*

$$\frac{W(\bar{A}(\mathbf{b}_{1:m});\mathbf{v}_{1:m})}{\max_{\sigma\in\mathcal{D}(T)} W(\sigma;\mathbf{v}_{1:m})} \geq \frac{\sum_{\mathsf{u}}\Omega_{\mathsf{u}}/\kappa_{\mathsf{u}}}{\sum_{\mathsf{u}}\Omega_{\mathsf{u}}} \geq \frac{1}{\max_{\mathsf{u}}\kappa_{\mathsf{u}}}.$$

Proof. Fix an internal node u and an unordered pair of children $\{\mathsf{c},\mathsf{d}\}$. Without loss of generality suppose that $\bar{v}(\mathsf{c}) \geq \bar{v}(\mathsf{d})$. The maximum welfare contribution of this pair is therefore $N(\mathsf{c})N(\mathsf{d})\bar{v}(\mathsf{c})$. We now compare this with the contribution under the average-bid order:

1. If $\bar{A}(\mathbf{b}_{1:m})$ places c before d, then the pair contributes exactly $N(\mathsf{c})N(\mathsf{d})\bar{v}(\mathsf{c})$, and so it obtains its maximum possible contribution.

2. Now suppose that $\bar{A}(\mathbf{b}_{1:m})$ places d before c. Because the children are ordered by descending average bid, $\bar{b}(\mathsf{d}) \geq \bar{b}(\mathsf{c})$. By Definition 4.3, we have that $\bar{b}(\mathsf{c}) = \alpha(\mathsf{c})\bar{v}(\mathsf{c})$ and $\bar{b}(\mathsf{d}) = \alpha(\mathsf{d})\bar{v}(\mathsf{d})$, and so $\alpha(\mathsf{d})\bar{v}(\mathsf{d}) \geq \alpha(\mathsf{c})\bar{v}(\mathsf{c})$. Rearranging terms, we have that

$$\frac{\bar{v}(\mathsf{d})}{\bar{v}(\mathsf{c})} \geq \frac{\alpha(\mathsf{c})}{\alpha(\mathsf{d})} \geq \frac{1}{\kappa_{\mathsf{u}}}$$

   and therefore the pair's welfare contribution is $N(\mathsf{c})N(\mathsf{d})\bar{v}(\mathsf{d}) \geq \frac{1}{\kappa_{\mathsf{u}}}N(\mathsf{c})N(\mathsf{d})\bar{v}(\mathsf{c})$.

Therefore, whether or not the average-bid order agrees with the average value order, every unordered pair of children of u obtains at least $1/\kappa_{\mathsf{u}}$ of its maximum possible welfare contribution. Summing over all unordered pairs and internal nodes, we get that

$$\begin{aligned}(n-1)W(\sigma;\mathbf{v}_{1:m}) &= \sum_{\mathsf{u}} \sum_{\mathsf{c},\mathsf{d}\in\mathcal{C}(\mathsf{u}),\mathsf{c}\prec_{\bar{A}(\mathbf{b}_{1:m})}\mathsf{d}} N(\mathsf{c})N(\mathsf{d})\bar{v}(\mathsf{c}) \\ &\geq \sum_{\mathsf{u}} \frac{1}{\kappa_{\mathsf{u}}} \sum_{\{\mathsf{c},\mathsf{d}\}\subseteq\mathcal{C}(\mathsf{u})} N(\mathsf{c})N(\mathsf{d})\max\{\bar{v}(\mathsf{c}),\bar{v}(\mathsf{d})\}\end{aligned}$$

by Equation 3. To obtain the final result, we can apply Equation 4 to get

$$\frac{W(\bar{A}(\mathbf{b}_{1:m});\mathbf{v}_{1:m})}{\max_{\sigma\in\mathcal{D}(T)} W(\sigma;\mathbf{v}_{1:m})} \geq \frac{\sum_{\mathsf{u}}\frac{1}{\kappa_{\mathsf{u}}}\sum_{\{\mathsf{c},\mathsf{d}\}\subseteq\mathcal{C}(\mathsf{u})} N(\mathsf{c})N(\mathsf{d})\max\{\bar{v}(\mathsf{c}),\bar{v}(\mathsf{d})\}}{\sum_{\mathsf{u}}\sum_{\{\mathsf{c},\mathsf{d}\}\subseteq\mathcal{C}(\mathsf{u})} N(\mathsf{c})N(\mathsf{d})\max\{\bar{v}(\mathsf{c}),\bar{v}(\mathsf{d})\}} \geq \frac{1}{\max_{\mathsf{u}}\kappa_{\mathsf{u}}}.$$

where $\Omega_{\mathsf{u}} \coloneqq \frac{1}{n-1}\sum_{\{\mathsf{c},\mathsf{d}\}\subseteq\mathcal{C}(\mathsf{u})} N(\mathsf{c})N(\mathsf{d})\max\{F_v(\mathsf{c})/N(\mathsf{c}), F_v(\mathsf{d})/N(\mathsf{d})\}$. □

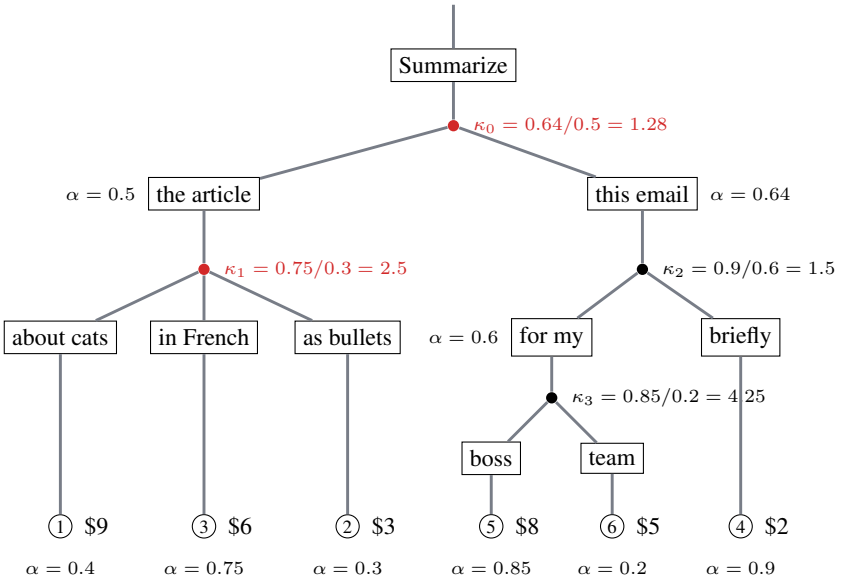


Figure 7: Welfare under heterogeneous pacing. Leaves show true values v_r and pacing factors α_r, so that $b_r = \alpha_r v_r$. Internal subtrees are labeled with their value-weighted pacing factor $\alpha(\mathsf{c}) = F_b(\mathsf{c})/F_v(\mathsf{c})$, and κ_{u} is the ratio of the largest to smallest $\alpha(\mathsf{c})$ among the children of u. Red nodes are where paced bids reverse the value-optimal order. At the root, "this email" now has the higher average bid (3.2 versus 3), and under "the article," request 3 now outbids request 1 (4.5 versus 3.6). The realized schedule is $5, 6, 4, 3, 1, 2$, which attains $87/99 \approx 0.88$ of the optimal DFS welfare. Theorem 4.4 guarantees at least $\sum_{\mathsf{u}} \Omega_{\mathsf{u}}/\kappa_{\mathsf{u}} / \sum_{\mathsf{u}} \Omega_{\mathsf{u}} \approx 0.63$, and hence at least $1/\max_{\mathsf{u}} \kappa_{\mathsf{u}} \approx 0.24$. Node 3 has the largest condition number but no reversal, so it loses nothing.

D Appendix for Section 5: Experiments

D.1 Setup details

We use SGLang v0.5.17[9] serving DeepSeek-R1-Distill-Qwen-32B in bf16 with the engine's defaults, except that we fix its static memory fraction at 0.85, which leaves a 23k-token KV cache. Prefix caching and LRU eviction are handled by SGLang. Every query decodes at most 128 tokens, so TTFT is dominated by prefill and caching. The queue in front of the server holds Q queries, splits them into Q/n batches, orders each batch with one of our algorithms, and sends them sequentially without pausing the engine. The cache is cleared once per experiment (not between batches). The queue sends about 20 queries per second and the engine serves 3 to 4, so the engine's own queue has a few hundred entries for the whole run; TTFT is almost entirely time spent waiting in the queue (in the order we sent). The queue tokenizes prompts with the model's tokenizer, so the tree it optimizes is the same one built by SGLang.

We ran the experiments on three H100 nodes with slightly varying hardware. Welfare and hit rate do not depend on the machine; TTFT does, so throughout the paper we only compare TTFTs measured on the same machine. The threshold, correlation, and long-tree experiments are from the second machine and the one-cluster experiment from the third; everything else is from the first.

Welfare is measured with respect to the final ordering of queries ranked by TTFT. It agrees with the welfare of the planned average-bid DFS order to within 1.3% for all results, so we report only the realized value. Raw welfare varies substantially from trial to trial because a heavy-tailed draw puts a different amount of value in the batch each time, and it moves every algorithm together (Figure 9), which is why we plot the percentage over $\frac{1}{2}\sum_r b_r$. Cache hit rate is defined as the number of cached prompt tokens divided by the total number of prompt tokens, as reported by SGLang. All completion times, when reported, record the time from the first send to the last completion of the batch. The queue itself is a Python prototype: for a batch of 1024 queries it computes the order in under a

[9] https://github.com/sgl-project/sglang

millisecond and the VCG payments in 0.3 s by naive per-user recomputation (Appendix D.10 has the fast version), while actually serving the batch takes on the order of minutes.

D.2 WORKLOADS

MIXSHARE mixes three kinds of queries: SQuAD questions, 32 per article, with the article as a shared prefix; GSM8K questions over one of 12 few-shot prefixes; and singletons that share only a short instruction. In MIXSHARE-P, $P\%$ of the queries belong to one of the shared prefix clusters. AGENT replays logged SWE-agent trajectories, where each call's prompt is the history up to that call, so call k is a prefix of call $k+1$. TOT replays tree-of-thought on GSM8K (3-ary thought trees of depth 3). TOT-LONG is the same, with a SQuAD article of 3-5k tokens as the root of every tree; since each thought is a single sentence, nearly every token of every prompt is the article, which is why its shared prefixes are so long and its unique suffixes so short. Table 2 gives the tree statistics of one $Q = 1024$ instance of each and Figure 8 the per-query distributions. Note the third column of the table: every workload but TOT exceeds the KV cache several times over.

workload	total tokens	distinct tokens	distinct / KV cache	shared fraction	median tokens per query: prompt	shared prefix	unique suffix	queries at interior nodes
stars	2.05M	781k	33.7	0.62	1086	381	57	0
chains	3.32M	437k	18.9	0.87	2854	2852	0	876
short trees	168k	23k	1.0	0.86	159	147	12	357
long trees	4.07M	124k	5.4	0.97	4143	4127	15	335

Table 2: Radix-tree statistics of one $Q = 1024$ instance of each workload. Distinct tokens is the total edge length of the tree; a query's shared prefix is its longest common prefix with any other query. Stars corresponds to MIXSHARE-50, chains corresponds to AGENT, short trees corresponds to TOT, and long trees corresponds to TOT-LONG.

D.3 BATCH SIZE AND SHARING FRACTION

Figure 9 gives the raw welfare and TTFT behind the first two columns of Figure 1. In the sharing sweep at $n = 256$, nothing can be cached at $P = 0$ and the algorithms differ only through the order itself. As P grows, the completion time of the two DFS orders falls from about 560 s to 330 s while that of BID-SORT and NO-SORT rises to 740 s. In the batch-size sweep on MIXSHARE-50, NO-SORT does not depend on n by construction, which is a useful consistency check; for DFS-BID, the hit rate goes from 0.23 at $n = 32$ to 0.60 at $n = Q$ and mean TTFT from 316 s to 160 s. The ratio of the best DFS welfare to the unconstrained optimum falls gently with sharing, from 0.83 at $P = 0$ to 0.78 at $P = 75\%$, as deeper trees hit the DFS restriction more often. While one might expect DFS-BID and BID-SORT to agree at $P = 0$, singletons in our dataset contain a very short shared instruction, causing us to serve one group before another.

D.4 DIFFERENT TREE TOPOLOGIES

Figures 10, 11, and 12 vary the batch size on different radix tree topologies: chains, short trees, and long trees. On chains at $n = Q$, DFS-BID beats DFS-UNIFORM by 16% at the same hit rate and TTFT, while BID-SORT wins welfare by 48% and pays with a 0.48 hit rate (versus DFS-BID's 0.75) and $2.4\times$ the TTFT. NO-SORT and BID-SORT have the same hit rate at every n, since sorting a random order by bid leaves it random from the cache's point of view. Short trees fit in the cache, so nothing is evicted. Here, every algorithm has the same 28 s TTFT. Long trees are where the DFS restriction of our inference auction has the largest impact. DFS-BID keeps a 0.96 hit rate and a 46 s TTFT, and BID-SORT drops to 0.20 and 559 s, because every branch it interleaves recomputes a 4k-token prefix.

D.5 CORRELATED VALUES

The main results draw bids i.i.d. per query, which is the case of many users sharing a prefix with unrelated values and the hardest case for DFS-BID. To interpolate toward the case where one user owns a whole cluster, we draw values from a Gaussian copula with within-cluster correlation ρ

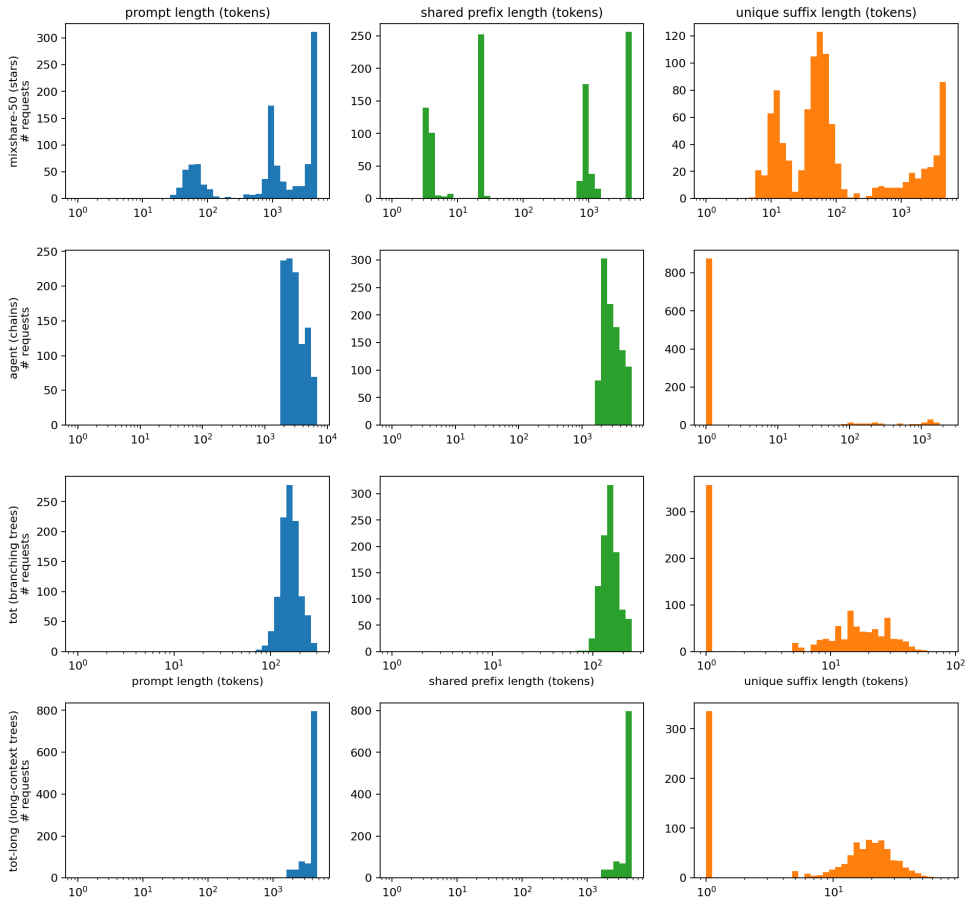


Figure 8: Per-query distributions of prompt, shared prefix, and unique suffix length (log scale).

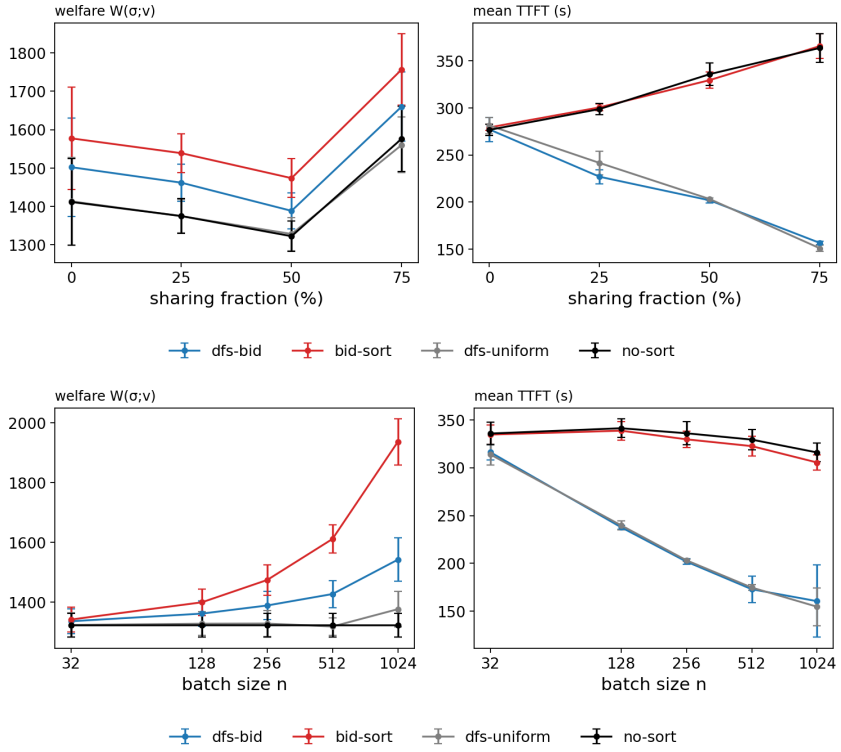


Figure 9: Raw welfare and mean TTFT for the sharing-fraction (top) and batch-size (bottom) sweeps of Figure 1.

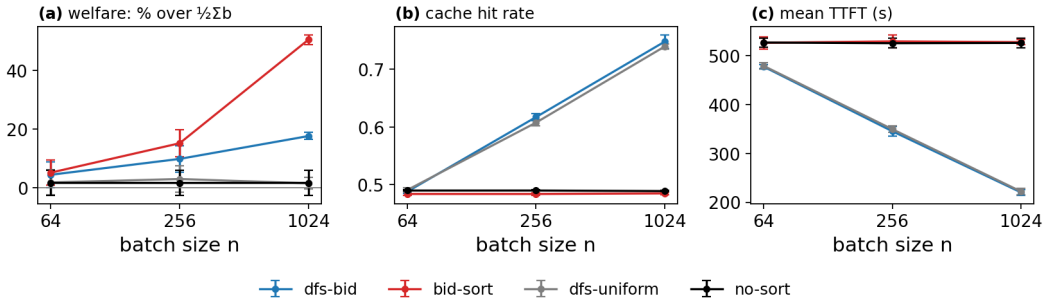


Figure 10: Chains (AGENT) against batch size n.

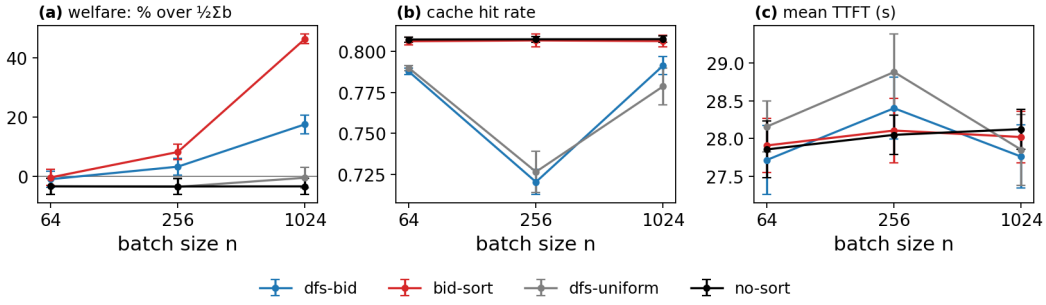


Figure 11: Short trees (TOT) against batch size n. The workload fits in the cache, so hit rate and TTFT do not depend on the order.

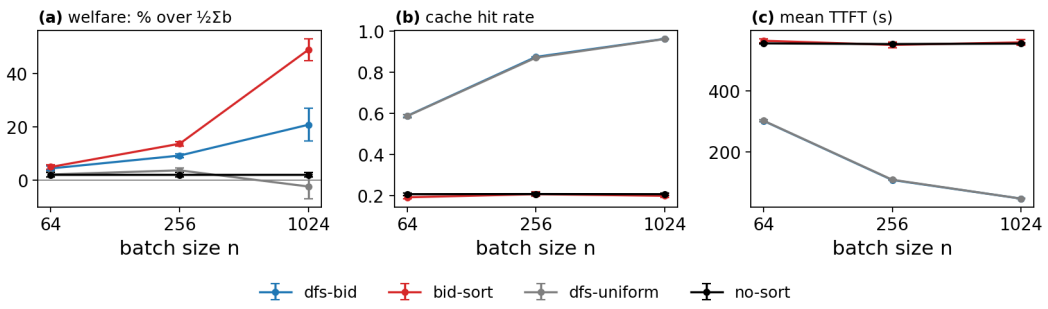


Figure 12: Long trees (TOT-LONG) against batch size n, second machine.

and the same Pareto marginal. A cluster is a set of queries sharing at least 128 prefix tokens on stars, a trajectory on chains, and a tree on trees; singletons are unaffected. Each cluster c draws $Z_c \sim \mathcal{N}(0,1)$ and each query r in it draws $\epsilon_r \sim \mathcal{N}(0,1)$; its value is $v_r = 1 + (1 - \Phi(z_r))^{-1/1.5}$ with $z_r = \sqrt{\rho}\, Z_c + \sqrt{1-\rho}\,\epsilon_r$. So $\rho = 0$ is i.i.d. and $\rho = 1$ gives every query in a cluster the same value.

Figure 13 plots welfare against ρ on stars and the right column of Figure 1 displays the engine results at $\rho \in \{0, 0.5, 1\}$. DFS-BID gains as ρ grows because the value-sorted order and the tree-contiguous order agree more often: the best DFS welfare rises from 0.83 to 0.91 of the unconstrained optimum. BID-SORT does not gain, and it still thrashes the cache at $\rho = 0.5$ (hit rate 0.29 against 0.60). At $\rho = 1$ its order happens to keep clusters together and its hit rate matches DFS-BID's, but DFS-BID is still faster (mean TTFT 85 s against 124 s), because it serves short high-value clusters first.

Figure 14 repeats this experiment on chains and long trees at $n = Q$. On long trees at $\rho = 1$ the two optima coincide: DFS-BID reaches 0.999 of the unconstrained welfare at the same hit rate and TTFT as BID-SORT. On chains it reaches 0.90, and the remaining gap is the cost of the DFS restriction when trajectories on the same task share interior nodes with other users. On both, BID-SORT recovers the cache only at $\rho = 1$; at $\rho = 0.5$ its TTFT is still $2\times$ DFS-BID's on chains and $10\times$ on long trees.

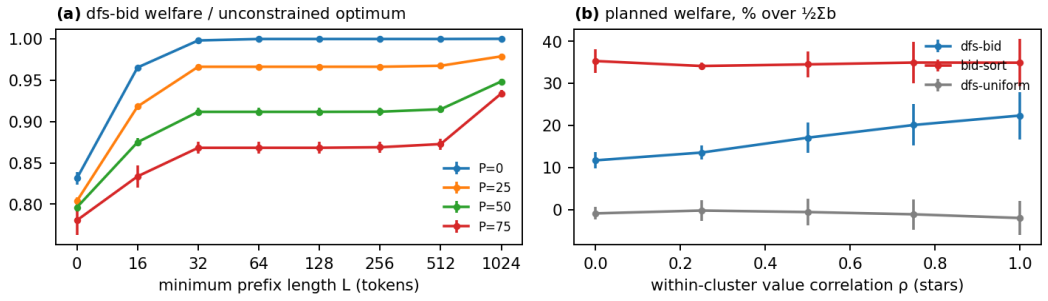


Figure 13: Offline sweeps on the planned orders, three trials. (a) DFS-BID's welfare as a fraction of the unconstrained optimum against the minimum-prefix threshold L of Appendix D.3, per sharing fraction. (b) Planned welfare of each algorithm against within-cluster value correlation ρ.

D.6 ONE CLUSTER BIDS

We consider now a situation similar to that of the highest-correlation setting of the experiments in Figures 13 and 14 (though in the original setting of i.i.d. Pareto bids). We draw one cluster of queries with a shared prefix at random among the clusters with at least eight members (32 to 58 queries on these instances). That cluster keeps its Pareto bids. Every other bid is set to zero (Figure 3).

This bid distribution is meant to model a realistic scenario where most users do not submit any bid for priority (such users still participate in the auction and must be served in a timely fashion).

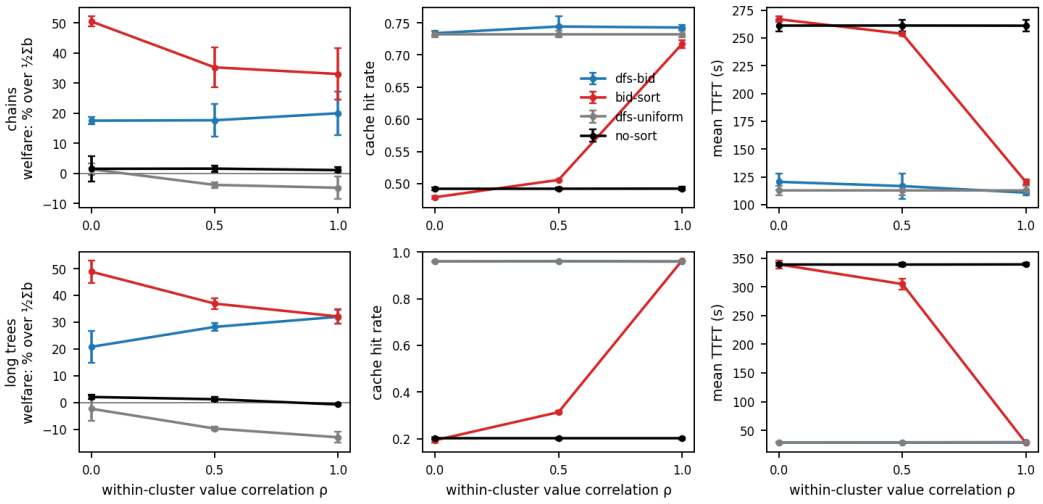


Figure 14: Within-cluster value correlation ρ on chains (top) and long trees (bottom) at $n = Q$, third machine.

Serving the cluster with non-zero bids first corresponds to both the unconstrained optimum and a DFS order, so DFS-BID matches BID-SORT's welfare at every n, where the paying cluster gets its first tokens in about a second under either algorithm. The margin over the bid-blind algorithms grows with n, from nothing at $n = 32$, where a batch rarely holds more than one or two members of the cluster, to about 100 points at $n = Q$. DFS-BID's hit rate and completion time equal DFS-UNIFORM's at every n, while BID-SORT's hit rate stays near 0.25 and its TTFT is about $1.5\times$ at every n above 32.

D.7 Online arrivals

The main protocol has the whole queue present at $t = 0$, so the batch-size sweep in Figure 1 shows the benefit of a larger batch but not its cost (the wait for the batch to fill). Here the 1024 queries of MIXSHARE-50 arrive as a Poisson process at 4 or 1 per second, a batch is released when its n-th query arrives, and TTFT is measured from arrival. NO-SORT is batched the same way, so the batching cost is common to all algorithms. Table 3 reports means over three trials.

rate	n	wait	DFS-BID mean TTFT	DFS-BID hit	NO-SORT mean TTFT
4/s	64	8	150	0.33	211
4/s	512	63	160	0.56	314
1/s	64	31	45	0.30	47
1/s	512	254	328	0.56	406

Table 3: Poisson arrivals on MIXSHARE-50. "Wait" is the mean time from a query's arrival to the release of its batch; all times in seconds.

At 4 queries per second the server is saturated: the queue arrives in about 250 s and takes over 500 s to serve. Going from $n = 64$ to $n = 512$ adds 55 s of batch wait but changes DFS-BID's mean TTFT by between -7 and $+29$ s across trials, because the added wait mostly replaces time the query would have spent waiting in the engine, while the hit rate rises from 0.33 to 0.56. The same change costs NO-SORT 100 s and buys it nothing. At 1 query per second the server keeps up with arrivals, there is no queue to hide the wait, and $n = 512$ costs about 280 s of TTFT. Under load a large batch is nearly free; under light load a timeout should release batches long before they fill.

D.8 SECOND MODEL

Table 4 repeats our main experiments with Qwen2.5-7B-Instruct (Qwen Team, 2025) in place of the 32B model, on the same instances and with the KV cache capped at the same number of tokens. The two models share a tokenizer, so the trees are identical. Welfare agrees to the unit in every experiment and hit rates within a few hundredths; every time metric is 4–5× shorter, uniformly across algorithms.

		welfare		hit rate		completion time (s)	
experiment	algorithm	7B	32B	7B	32B	7B	32B
MIXSHARE-50, $n = Q$	DFS-BID	1541	1542	0.60	0.60	87	319
	BID-SORT	1936	1936	0.22	0.23	163	615
	DFS-UNIFORM	1376	1376	0.60	0.59	91	322
	NO-SORT	1322	1322	0.21	0.21	165	629
MIXSHARE-75, $n = 256$	DFS-BID	1660	1660	0.70	0.69	81	327
	BID-SORT	1757	1757	0.29	0.28	188	739
AGENT, $n = Q$	DFS-BID	1853	1854	0.82	0.75	77	438
	BID-SORT	2375	2375	0.49	0.48	227	1076
TOT, $n = Q$	DFS-BID	1686	1686	0.86	0.79	54	61
	BID-SORT	2098	2098	0.78	0.81	53	60

Table 4: Qwen2.5-7B-Instruct against the 32B model on identical instances; means over three trials.

D.9 OTHER VALUE DISTRIBUTIONS

Figure 15 repeats the MIXSHARE-50 and chains experiments at $n = Q$ with bids pulled from different distributions, namely uniform and lognormal. The cache hit rate and TTFT does not change, since they depend on the tree and not on the bids. DFS-BID's gain over DFS-UNIFORM is positive and significant under every distribution and grows with the tail, as does BID-SORT's gain over DFS-BID.

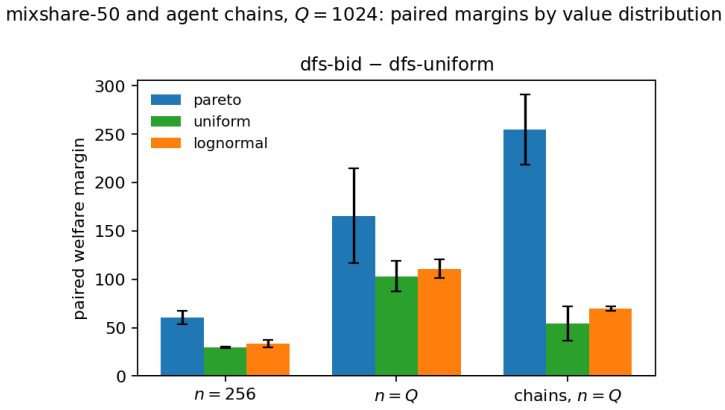


Figure 15: Welfare gain on MIXSHARE-50 and AGENT under Pareto, uniform on $[1, 5]$, and lognormal ($\sigma = 0.5$) values with the same mean.

D.10 FAST VCG PAYMENTS

The experiments compute payments by rerunning the allocation once per user, which incurs a quadratic run-time. We also implemented the algorithm of Theorem 3.3, which re-sorts only the affected children on a user's root-to-leaf paths. Its payments agree with the naive computation to 10^{-12} on random trees and on our stored instances, and at $n = 1024$ it takes 34 ms while the naive algorithm takes 370 ms on average.

D.11 AUTOBIDDING

We simulate Algorithm 3 over $T = 60$ batches of $n = 256$. For stars, each shared-prefix cluster belongs to a distinct user. For chains, each trajectory belongs to a distinct user. Half the users pace toward a target spend of half their mean truthful payment with step size $\eta = 0.1$ normalized by the target; the rest bid truthfully. Figure 16 shows the usual behavior of dual descent with a fixed step: spend runs above the target while the multiplier is small and below it once the multiplier has grown. Welfare under pacing averages 0.91 of the DFS optimum at true values on stars and 0.93 on chains, and the bound of Theorem 4.4 holds in every batch.

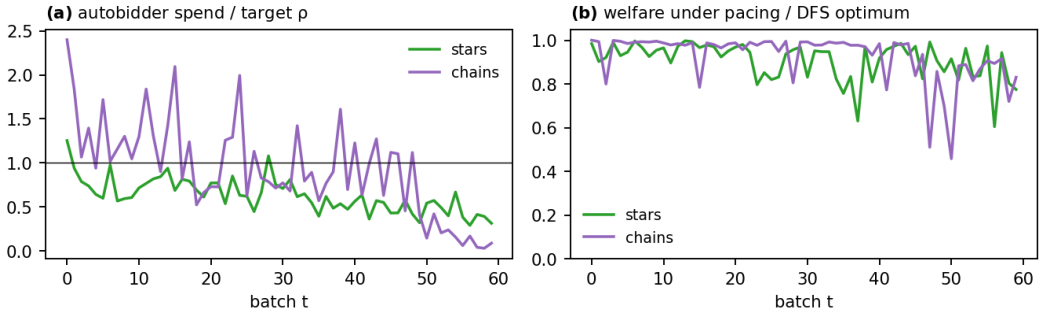


Figure 16: Pacing autobidders over $T = 60$ batches of $n = 256$, half the users pacing and half truthful. (a) Spend relative to the per-batch target. (b) Welfare under pacing as a fraction of the DFS optimum at true values.