Solver-Backed Neural Combinatorial Auctions: Exact Feasibility, Auditable Allocations, and Regret-Stability Bounds

Liz Lemma Future Detective

January 16, 2026

Abstract

Differentiable economics (e.g., RegretNet) learns revenue-maximizing auctions by gradient methods but struggles in combinatorial auctions because feasibility is hard to enforce and randomization/relaxations can be difficult to implement. Recent CA-focused architectures (CANet/CAFormer) enforce feasibility via differentiable constructions (softmax/min factorization), enabling randomized mechanisms but leaving a gap between differentiable feasibility and operational implementability, and introducing optimization brittleness. We propose a neuro-symbolic alternative: a neural network proposes solver-friendly bundle scores (and optional price parameters), and an exact winner-determination optimizer (MIP/DP) computes the allocation that maximizes these scores subject to strict combinatorial feasibility. Payments are learned with an IR-by-design scaling, and incentive compatibility is enforced approximately via regret minimization as in RegretNet, now with the allocation guaranteed feasible and integral at deployment. Our core contributions are (i) an exact feasibility/implementability guarantee by construction, (ii) a clean IR guarantee, and (iii) a stability bound quantifying how solver suboptimality and regret-oracle error translate into true regret. We benchmark against CAFormer/CANet and heuristic AMA/VVCA baselines, showing improved robustness and auditable deployment pathways in 2026-style settings with sophisticated bidders and operational constraints.

Table of Contents

3. 3. Model: combinatorial auction primitives, bundle space and bidding language, feasibility constraints, utilities, regret and IR definitions; discussion of what is closed-form vs what is numerical.

4. 4. Mechanism Class (Solver-Backed): score network, exact winner determination objective, tie-breaking and measurability, payment parameterization with IR-by-design; optional contextual features and permutation-equivariance.

5. 5. Training as Bilevel Optimization: outer objective (revenue with regret penalty/budget); inner deviation search; three gradient strategies (LP relaxation + implicit differentiation; straight-through estimators; primal-dual / cutting-plane surrogates); when each is appropriate.

6. 6. Theory: (i) feasibility/implementability guarantee, (ii) ex-post IR guarantee, (iii) regret stability bound under solver/oracle approximation, (iv) discussion of integrality gap when training on relaxations and deploying integral solutions.

7. 7. Experiments: synthetic CAs (2x2, 2x3, 2x5) and scaling studies; compare to CAFormer/CANet and heuristic baselines; stress-test regret with stronger deviation oracles; runtime and solver statistics; ablations on solver tolerance and gradient method.

8. 8. Auditing and Deployment: reproducibility of allocations, logging solver certificates, transparency of score functions, handling randomized extensions (optional), and governance constraints.

9. 9. Conclusion and Next Steps: open problems (large m with bidding languages, stronger IC certificates, online updates under shift) and how solver-backed pipelines fit platform realities.

1 Introduction

Combinatorial allocation problems sit at the center of modern market design, but the engineering requirements of deployment have sharpened in ways that are easy to understate. In 2026, large marketplaces and procurement platforms routinely operate under contractual and regulatory constraints that treat feasibility as a hard requirement rather than an average-case aspiration: an item cannot be allocated twice; a bidder cannot be assigned two incompatible bundles; and an outcome must be auditable as a deterministic mapping from reported bids to allocations and payments. When these constraints fail even rarely, the consequences are not merely small welfare losses. They include downstream operational failures (e.g., infeasible fulfillment plans), legal exposure (e.g., breach of non-discrimination commitments), and strategic vulnerabilities (e.g., bidders learning to induce pathological rounding). This paper takes those deployment realities as the starting point, and asks how we can leverage modern learning architectures without softening the core implementability guarantees that combinatorial auctions require.

The appeal of learned mechanisms is clear. Rich bidder preferences over bundles generate a high-dimensional input space, and classical parametric families can be too rigid to capture the allocation and pricing patterns that maximize revenue subject to reasonable incentive constraints. Neural networks offer a flexible way to map bid profiles to allocation-relevant signals and pricing parameters. The central difficulty is that the economically meaningful output—a feasible allocation of indivisible items—is intrinsically discrete. Most end-to-end differentiable approaches therefore introduce a surrogate: they relax the combinatorial feasibility constraints, compute a fractional or smoothed assignment, and then either (i) interpret the relaxed output as a lottery, or (ii) round it to a deterministic allocation. These approaches are often attractive during training because they supply gradients, but they shift the key economic question to the gap between the surrogate and the deployed mechanism.

This gap is not benign. A "soft feasibility layer" can satisfy constraints only approximately, in expectation, or up to numerical tolerance, yet a deployed marketplace must satisfy feasibility exactly on every instance. Even if one rounds relaxed allocations, the resulting outcome can be discontinuous in bids, can violate monotonicity properties that support incentive alignment, and can interact unpredictably with payment rules that were tuned to the relaxed model. Put differently, the relaxation may be differentiable, but the deployed mechanism is not the one being optimized. From an economic perspective, this breaks the link between the training objective (often a proxy for revenue subject to a proxy for incentive compatibility) and the strategic object of interest (utility under the actual integer allocation rule). From an operational perspective, it invites a fragile pipeline in which a small change in solver tolerances, hardware, or rounding heuristics can alter allocations

and payments in ways that are hard to certify or explain.

We therefore focus on a solver-backed design principle: feasibility should be enforced by construction at the point of deployment, using the same constraints that define the intended combinatorial auction. Concretely, we let a neural scorer map the bid profile to a matrix of scores over bidder–bundle pairs, and we then select an allocation by solving a winner-determination problem over the true feasible set. The optimizer is not an approximation to feasibility; it *is* feasibility. This architecture matches how many real platforms already operate: they have mature integer-optimization pipelines (often with years of tuning and domain-specific constraints), and the question is how to incorporate learning into the decision rule without discarding the solver that guarantees correctness. The neural network does not output an allocation directly; it outputs guidance to an allocation solver that remains the final arbiter of implementability.

A second deployment-driven requirement concerns individual rationality (IR). In many applications, especially where participants are businesses with outside options, ex-post IR is not merely a desirable equilibrium property; it is a participation constraint that must hold mechanically to avoid disputes. Standard payment constructions that target incentive compatibility (e.g., VCG-style payments) can be computationally expensive or incompatible with the chosen allocation rule, and learned payment networks can inadvertently produce negative utilities for truthful bidders if unconstrained. We adopt a simple IR-by-design parameterization: each bidder's payment is a scale factor in $[0, 1]$ applied to her own reported bid for the allocated bundle. This guarantees that a truthful bidder never pays more than her value for what she receives, independent of others' bids. The point is not that this payment form is universally optimal, but that it provides a clean baseline that is robust to training error and easy to audit.

Our third motivating concern is incentive robustness under approximation. Even when feasibility is exact, learned mechanisms are typically evaluated and trained using approximate best-response computations, and real winner-determination solvers may return near-optimal solutions with a known optimality gap. Both approximations matter strategically. If the regret oracle fails to find profitable deviations, a mechanism can appear approximately incentive compatible while still being exploitable by sophisticated bidders. If the allocation solver is slightly suboptimal in the score objective, the induced allocation can change discontinuously, and the corresponding utilities can shift in ways that create artificial regret. Deployment again forces us to be explicit: we need to understand how these approximation errors propagate into the economic quantities we care about, rather than implicitly assuming away the gap between an idealized model and the implemented system.

Within this framing, the contribution of the paper is to articulate and analyze a mechanism class in which the hardest deployment constraint—combinatorial implementability—is handled exactly, while the remaining

learning components are designed to admit transparent guarantees and error accounting. We organize the analysis around three claims. First, because the winner-determination step optimizes over the true combinatorial feasibility set, the deployed allocation is always deterministic and feasible, avoiding lottery interpretations and eliminating the need for decomposition or rounding arguments. Second, because payments are constrained through an explicit scaling factor, truthful participation is ex-post individually rational by construction. Third, when we relax the idealizations used in training—allowing an η-optimal allocation solver and a δ-approximate deviation oracle—we can bound the gap between measured regret and true regret under mild regularity (Lipschitz) conditions on how payments respond to allocation changes. This last point is conceptually important: it separates what is genuinely learned (the scoring and scaling functions) from what is guaranteed by engineering modules (the solver) and clarifies which approximations can cause incentive violations.

Beyond these headline properties, we view the solver-backed lens as a way to discipline the broader conversation about differentiability in market design. Differentiable surrogates are invaluable during training, but the correct benchmark is the deployed mechanism. When training differentiates through a continuous relaxation of the winner-determination problem, one is effectively optimizing a different allocation correspondence than the one implemented at test time. The resulting training–deployment mismatch can show up as revenue loss, increased regret, or brittle sensitivity to instance distribution. We therefore treat the relaxation not as part of the mechanism definition, but as a numerical tool whose error can be empirically measured and, under suitable smoothness, theoretically related to performance differences. This perspective supports a pragmatic workflow: keep deployment exact and auditable, use relaxations only where they provide tractable gradients, and explicitly track how far the relaxed solution is from the integral one on the instances that matter.

We also emphasize what this approach does *not* solve. Solver-backed implementability does not make the mechanism strategyproof; it simply ensures that whatever strategic properties we achieve are properties of a well-defined, feasible, deterministic auction. The payment scaling rule enforces truthful IR but can restrict revenue relative to more complex pricing rules, and it does not preclude profitable misreports. Moreover, the approach inherits the computational complexity of winner determination over the chosen bundle language; when K is large, the solver can become the bottleneck, and careful instance design (or restricted bidding languages) remains essential. Finally, our guarantees are conditional on bounded values and on regularity properties that may fail for highly discontinuous payment schemes; learning can still produce sharp decision boundaries that are strategically fragile under distribution shift.

Taken together, the message is simple but, we think, underappreciated:

in combinatorial auctions, feasibility and implementability are the non-negotiable substrate on which all other economic objectives sit. A mechanism that is elegant in a relaxed model but operationally infeasible is not merely imperfect; it is not deployable. By structuring the mechanism around an exact solver and embedding learning only in components that do not compromise feasibility, we obtain a clean separation between what must be guaranteed and what can be optimized. This separation illuminates the central trade-off we study throughout: how far one can push revenue via expressive learned scoring and pricing while maintaining robust, quantifiable control over incentive problems under the approximations that real systems necessarily use.

2 Related Work

Our work sits at the intersection of (i) learning-based mechanism design, where the mapping from reported types to outcomes is parameterized by a neural network and trained against incentive constraints, and (ii) solver-in-the-loop decision systems, where a learned model produces a score or cost vector that is then optimized subject to hard combinatorial constraints. The common thread is a shift in emphasis from closed-form auction rules to *implementable pipelines* that can be deployed in large marketplaces with complex constraints. The key difference across strands is where feasibility and incentive properties are enforced: directly by economic structure (e.g., VCG/affine maximizers), by differentiable relaxations and penalties (common in end-to-end learning), or by an external optimizer that enforces constraints exactly (the approach we pursue).

2.1 Differentiable mechanism design and regret-based training

A prominent modern line of work treats mechanism design as a learning problem: parameterize an allocation rule and payment rule by neural networks and train parameters to maximize expected revenue subject to approximate incentive compatibility, often operationalized by ex-post regret. RegretNet **?** is the canonical example in multi-item settings: it uses a differentiable architecture to produce allocations (frequently via softmax-like normalization or other continuous proxies) and payments, and it penalizes estimated regret computed by an inner optimization over misreports. Subsequent work refined the approach along several dimensions, including more expressive architectures, improved regret estimation, and better handling of constraints such as individual rationality or budget balance **??**. The broad appeal is conceptual and practical: rather than deriving a mechanism from first principles, one can search a rich function class while keeping incentives "in the loop" via adversarial deviations.

Two design challenges recur in this literature. First, allocation feasibility is often represented through a continuous relaxation (e.g., fractional allocations), followed by rounding or lottery interpretations. This can be entirely reasonable in divisible-goods environments, but becomes delicate in combinatorial auctions over indivisible items because feasibility is inherently discrete. Second, incentive constraints are evaluated through approximate best responses, typically using gradient-based inner loops; this creates a natural gap between the *measured* regret (against the deviation algorithm) and the *true* regret (against all misreports). Many papers acknowledge this gap and validate empirically with stronger attackers, but the conceptual point remains: approximate regret minimization is only as reliable as its deviation oracle, and the oracle itself can be sensitive to nonconvexities induced by the learned mechanism.

Our perspective is closest in spirit to the regret-minimization program, but we separate what is learned from what is guaranteed. In particular, we keep combinatorial feasibility outside the neural network by making the deployed allocation the outcome of a winner-determination solver over the true feasibility set. This shifts the learning burden from "learn to allocate feasibly" to "learn scores that induce good allocations when optimized exactly," which better matches how production marketplaces already enforce constraints. At the same time, we treat approximate best responses and solver tolerances as first-class objects, rather than hidden numerical details, which motivates explicit error accounting for incentive metrics.

2.2 Symmetry, equivariance, and transformer variants

A second cluster of related work builds inductive bias into learned mechanisms by exploiting the symmetries of auction environments: bidder identities are often exchangeable, and item labels can be permuted without changing the underlying economic structure. Architectures that are permutation-invariant or permutation-equivariant can therefore improve sample efficiency and stability. This includes deep set constructions, graph neural networks, and transformer-style models tailored to set inputs **??**. In auction settings, such inductive biases have been used both for allocation/payment networks and for value/policy prediction subroutines that feed into an optimizer.

In combinatorial auctions, symmetry-aware architectures are particularly compelling because the input dimensionality grows quickly with the bundle language. A well-designed equivariant scorer can share statistical strength across bidder–bundle pairs and can mitigate overfitting to arbitrary indexing conventions. However, symmetry alone does not resolve the central discreteness issue: even an equivariant network typically outputs continuous scores or fractional assignments, and one must still address how an integral allocation is produced and how the learning signal reflects that integral decision. Our solver-backed approach is complementary: permutation-

equivariant scoring can be layered on top of the exact winner-determination module, yielding an architecture that is both structurally aligned with the environment and operationally constrained by hard feasibility.

2.3 Combinatorial-auction-specific feasibility layers (CANet, CAFormer, and related approaches)

Several recent papers focus directly on combinatorial auctions and attempt to reconcile learning with feasibility by embedding a *feasibility layer* or structured decoder. Representative approaches (often associated with names such as CANet and CAFormer in the applied literature) use architectures that output bundle-level assignment scores and then produce allocations via differentiable approximations to winner determination, or via specialized rounding procedures designed to satisfy non-overlap constraints. Some methods employ LP relaxations of the underlying integer program and differentiate through the relaxed KKT system; others use continuous surrogates such as Sinkhorn balancing, Gumbel perturbations, or soft matching layers to approximate combinatorial structure while retaining gradients.

These designs are valuable because they highlight the central engineering constraint: winner determination is not a minor detail, but the core of the mechanism. The feasibility-layer view also aligns with how practitioners implement allocation constraints (often as optimization models maintained by operations teams). The limitation, from our standpoint, is that a feasibility layer is often *trained* as a relaxation and then *deployed* after discretization. The resulting training–deployment mismatch can be material: the model may learn to exploit artifacts of the relaxed problem (e.g., spreading mass across bundles) that disappear after rounding, and payment rules tuned on relaxed outputs can behave unpredictably when the final integer allocation changes discontinuously.

Our design choice is therefore intentionally conservative: we treat feasibility as a hard constraint that is enforced at deployment exactly by an integer optimizer. Relaxations, when used, are treated as numerical tools for training rather than part of the mechanism definition. This mirrors a common practice in industrial systems: the optimization model is a stable "contract" that is auditable and version-controlled, while the learned components provide guidance (scores, priors, or warm starts) without changing the feasibility specification.

2.4 Classical truthful mechanisms: VCG, affine maximizers, AMAs, and VVCAs

A different but essential reference point is the classical mechanism design literature, which provides exact incentive guarantees through carefully structured allocation and payment rules. In combinatorial auctions with quasilin-

ear utilities, the Vickrey–Clarke–Groves (VCG) mechanism achieves dominant-strategy incentive compatibility (DSIC) and efficiency when the allocation maximizes reported welfare **?**. More generally, affine maximizers (sometimes framed as maximal-in-range or weighted welfare maximization with bundle-dependent offsets) paired with VCG-style payments preserve DSIC while restricting the allocation rule to a fixed range or a transformed objective **?**. This is closely related to the literature on virtual valuations and Myersonian approaches in multi-dimensional settings, though full optimality typically requires complex type spaces and is computationally challenging in combinatorial domains.

Approximate mechanism design (AMD) and approximately optimal mechanisms (often abbreviated informally as AMAs) explore how to trade off optimality and tractability, including welfare approximations and restricted message spaces **?**. More recent work on generalized VCG variants and virtual VCG auctions (VVCAs) seeks to retain incentive properties under certain parametric transformations of bids or values, sometimes motivated by the desire to capture revenue-optimal distortions while keeping computation manageable. These lines underscore an important point: if one can commit to an affine-maximizer allocation rule and compute the corresponding VCG payments (including counterfactual solves), then exact DSIC is available.

We view these truthful constructions as both a benchmark and a design alternative. They offer strong guarantees but can be restrictive in practice: (i) the required payments may be computationally heavy in large combinatorial problems due to multiple counterfactual optimizations; (ii) the space of affine transformations may be too rigid to match revenue goals under realistic value distributions; and (iii) platforms sometimes face business constraints (e.g., capped payments, posted-price-like requirements, or auditability constraints on pricing formulas) that complicate direct use of VCG. Our mechanism class can be seen as occupying a different point in the design space: we retain exact feasibility and a simple, auditable IR guarantee, while accepting that incentive compatibility is approximate and must be measured and controlled.

2.5 Differentiable optimization layers and decision-focused learning

Finally, our solver-backed architecture connects to a growing machine learning literature on differentiable optimization and decision-focused learning. Methods such as OptNet, differentiable convex optimization layers, and implicit differentiation through KKT conditions make it possible to embed linear or convex programs inside neural networks and backpropagate through their solutions **??**. More recent work develops gradient estimators for discrete optimization via perturb-and-MAP, straight-through estimators, and continuous relaxations, as well as frameworks that train predictive models

end-to-end for downstream decisions ("predict-then-optimize" and SPO-style objectives) **??**.

The key lesson from this literature is methodological: one can often train a model to be useful for an optimizer without requiring the entire pipeline to be smooth everywhere. However, discrete solvers introduce nondifferentiabilities and potential instability, and gradient surrogates can bias learning toward the relaxed problem. This resonates directly with our deployment motivation. In high-stakes allocation systems, the solver is not merely a training artifact; it is the operational engine that must satisfy hard constraints and be robust to corner cases. We therefore adopt a solver-in-the-loop stance that is common in decision-focused learning, but we evaluate it through an economic lens: the relevant objects are utilities, revenue, and regret under the *deployed* integral allocation rule and the *implemented* payments.

2.6 Positioning and roadmap

To summarize, existing learning-based mechanism design work provides powerful tools for searching over expressive outcome rules, but often relies on relaxed feasibility and approximate incentive enforcement. Classical truthful mechanisms provide exact DSIC but can be computationally burdensome or too rigid for revenue optimization in rich combinatorial domains. Differentiable optimization offers a pragmatic bridge, yet it raises the question of what guarantees survive discretization and solver tolerances.

Our contribution is to organize these themes around a deployment-first separation: exact feasibility is handled by the integer optimizer; individual rationality is enforced by a transparent payment parameterization; and incentive robustness is assessed with explicit accounting for solver and oracle approximations. With this positioning in place, we now turn to the model, where we formalize the combinatorial auction primitives, the bidding language, the feasibility constraints, and the regret and IR criteria that we use throughout.

3 Model

We study a single-shot combinatorial auction with a finite set of bidders N (with $|N| = n$) and a finite set of indivisible items M (with $|M| = m$). An *allocation* assigns to each bidder at most one *bundle* of items and ensures that no item is allocated to more than one bidder. Our goal in this section is to make the primitive objects explicit—the bundle language, feasibility constraints, and preference and reporting spaces—and to state the incentive and participation criteria that we use to evaluate learned mechanisms. We also highlight which parts of the framework are closed-form (and therefore

exact by construction) versus which parts rely on numerical optimization in practice.

3.1 Bundles and bidding language

Let $K \subseteq 2^M \setminus \{\emptyset\}$ denote the collection of *allowable bundles* over which bidders may express values and bids, with $|K| = k$. We write $S \in K$ for a generic bundle. Allowing an arbitrary K captures the practical reality that a platform often restricts the message space: full enumeration of 2^M is infeasible except for very small m, and many marketplaces use structured bundle languages (e.g., only singletons and a curated set of packages). Our analysis treats K as fixed and common knowledge.

Each bidder $i \in N$ has a (private) valuation vector

$$v_i = (v_{iS})_{S \in K} \in [0, V_{\max}]^K,$$

where v_{iS} is the value of receiving bundle S. We impose a uniform upper bound $V_{\max} < \infty$ for two reasons. First, it is consistent with the bounded supports used in training (values are typically normalized). Second, boundedness is convenient for stability and error accounting when we later discuss approximate solution concepts and numerical approximations.

We emphasize that K serves two conceptually distinct roles. Economically, it defines which outcomes the mechanism may assign to a bidder (because we will allocate at most one $S \in K$ per bidder). Operationally, it defines which reports bidders can submit. When K is rich (e.g., all nonempty bundles), the model approximates an unrestricted combinatorial auction. When K is sparse, the mechanism is best interpreted as operating under a restricted language: bidders may still have underlying preferences over all subsets of M, but only the coordinates in K are elicited and acted upon. In that case, incentive properties we state are understood relative to this restricted report space.

We assume an independent private values environment and take the valuation profile $v = (v_1, \ldots, v_n)$ to be drawn from a distribution $\mathcal{D}$ over $[0, V_{\max}]^{n \times K}$. The distribution $\mathcal{D}$ is used to define expected performance metrics (revenue and regret) and to motivate the training objective; the mechanism itself is defined pointwise for every realized report profile.

3.2 Reports, allocations, and feasibility

In the direct mechanism we study, each bidder submits a bid vector

$$b_i = (b_{iS})_{S \in K} \in \mathbb{R}_+^K,$$

and we write $b = (b_1, \ldots, b_n)$ for the bid profile. For analysis and for numerical training, it is often convenient to restrict bids to a compact set such

as $\mathcal{B}_i = [0, V_{\max}]^K$, either by design (bid caps) or by clipping. We keep the notation $\mathcal{B}_i$ for an admissible bid space when needed and otherwise treat bids as nonnegative.

We represent deterministic allocations using binary indicators $x_{iS} \in \{0,1\}$, where $x_{iS} = 1$ means bidder i receives bundle S. Let $x \in \{0,1\}^{n \times k}$ be the full allocation matrix. Feasibility is encoded by the standard combinatorial auction constraints:

$$\sum_{i \in N} \sum_{S \in K:\, j \in S} x_{iS} \leq 1 \qquad \forall j \in M, \tag{1}$$

$$\sum_{S \in K} x_{iS} \leq 1 \qquad \forall i \in N. \tag{2}$$

Constraint (1) enforces that each item is allocated at most once across all bidders, and (2) enforces that each bidder receives at most one allowable bundle. We denote the resulting feasible set by

$$\mathcal{X} \;=\; \Big\{ x \in \{0,1\}^{n \times k} :\ \text{(1) and (2) hold} \Big\}.$$

A (deterministic) allocation rule is a mapping $x(\cdot)$ that assigns to each bid profile b an element $x(b) \in \mathcal{X}$.

This binary formulation makes the discrete nature of the problem explicit. In particular, even for moderate m and expressive K, optimizing over $\mathcal{X}$ is NP-hard (winner determination). This computational fact is central for us: it motivates why, in deployment, a platform typically uses an exact mixed-integer optimizer or a high-quality heuristic. It also explains why many learning-based approaches introduce continuous relaxations during training. We will later separate these concerns by defining the *mechanism* using the deployed (integral) allocation, while treating any relaxation as a training instrument rather than as part of the economic object.

3.3 Outcomes, utilities, and payments

A direct mechanism maps bids to an outcome $(x(b), p(b))$, where $x(b) \in \mathcal{X}$ is the allocation and $p(b) = (p_1(b), \ldots, p_n(b)) \in \mathbb{R}^n_+$ is the payment vector charged to bidders. We assume quasi-linear utilities: for bidder i,

$$u_i(v_i; b) \;=\; \sum_{S \in K} x_{iS}(b)\, v_{iS} \;-\; p_i(b). \tag{3}$$

Because $x(b)$ assigns at most one bundle to each bidder, the value term in (3) simply picks out the value of the awarded bundle (or 0 if no bundle is awarded).

We evaluate mechanisms using both bidder-facing and seller-facing criteria. The seller's (gross) revenue at bid profile b is

$$\mathrm{rev}(b) \;=\; \sum_{i \in N} p_i(b).$$

Revenue is the natural objective for a platform, but it is only meaningful when paired with participation and incentive constraints: a high-revenue rule that induces systematic misreporting or violates individual rationality is not implementable in practice.

3.4 Incentives and participation

We use dominant-strategy incentive compatibility (DSIC) as a benchmark and ex-post regret as the operational measure of approximate incentive alignment. DSIC requires that truthful bidding is optimal regardless of what others report:

$$u_i\big(v_i;(v_i,b_{-i})\big) \;\geq\; u_i\big(v_i;(b_i',b_{-i})\big) \qquad \forall i,\ \forall v_i,\ \forall b_{-i},\ \forall b_i' \in \mathcal{B}_i. \tag{4}$$

Exact DSIC is generally demanding in combinatorial settings, especially when one seeks computational tractability and revenue optimization simultaneously. Consequently, much of the learning-based literature (and our approach) evaluates incentive alignment through *ex-post regret*, which captures the gain from the best unilateral deviation after seeing realized values.

Formally, for a fixed mechanism (and later, fixed parameters θ), define bidder i's ex-post regret at a valuation profile v as

$$\mathrm{rgt}_i(v) \;=\; \max_{b_i' \in \mathcal{B}_i} u_i\big(v_i;(b_i',v_{-i})\big) \;-\; u_i\big(v_i;(v_i,v_{-i})\big).$$

Taking expectation over $v \sim \mathcal{D}$ yields the expected ex-post regret

$$\mathrm{rgt}_i \;=\; \mathbb{E}_{v\sim\mathcal{D}}\big[\mathrm{rgt}_i(v)\big]. \tag{5}$$

A mechanism is approximately DSIC in the regret sense when rgt_i is small for all bidders. We stress an interpretive point: regret is an *ex-post* concept, so it does not require equilibrium assumptions. It is therefore well suited to empirical evaluation and to the adversarial-training paradigm, where a deviation algorithm searches for profitable misreports.

We also impose ex-post individual rationality (IR) under truthful bidding:

$$u_i\big(v_i;(v_i,b_{-i})\big) \;\geq\; 0 \qquad \forall i,\ \forall v_i,\ \forall b_{-i}. \tag{6}$$

IR matters both normatively (participants should not be harmed by truthful participation) and operationally (without IR, bidders may opt out or shade systematically). In our mechanism class, IR will be enforced by a simple payment parameterization; here, we keep IR as a criterion that any candidate mechanism must satisfy.

3.5 Closed-form structure versus numerical components

The model above is purely definitional: it specifies the environment and the desiderata. Implementing and *evaluating* a learned mechanism, however, introduces two computational problems that are easy to conflate but conceptually distinct.

First, *winner determination* is the optimization problem implicit in computing an allocation in $\mathcal{X}$. Even when the mechanism is defined as maximizing some objective over $\mathcal{X}$, solving that problem exactly can be expensive, and practical solvers often expose an optimality tolerance. This matters economically because allocation errors can translate into utility and revenue errors, and therefore into mismeasured regret. For this reason, when we later state bounds, we explicitly track a solver suboptimality level (denoted η) and interpret it as a property of the deployed optimization pipeline.

Second, *regret computation* itself is an optimization problem: the maximization over b_i' in (5) is rarely available in closed form for expressive mechanisms, especially when allocations change discontinuously with bids. In practice, one uses a deviation oracle (e.g., gradient ascent on bids, multi-start search, or other adversarial routines) that returns an approximately optimal deviation. We will denote its suboptimality by δ, reflecting the fact that the measured regret can underestimate true regret if the attacker fails to find the best misreport.

By contrast, two components of our framework will be deliberately *closed-form and exact*. The first is feasibility: allocations are required to lie in $\mathcal{X}$ as defined by (1)–(2). The second is the IR constraint (6), which we will guarantee through a payment form that never charges a bidder more than a (scaled) version of her own report for the allocated bundle. These exact properties are not merely theoretical conveniences; they correspond to features that are straightforward to audit and enforce in production systems.

This separation clarifies what our learning problem is *not*. We are not asking a neural network to learn the feasibility constraints of a combinatorial auction, nor are we relying on a relaxed allocation output that must later be rounded with unclear economic interpretation. Instead, the learned component will interact with the model through quantities that are compatible with an exact solver, while the unavoidable numerical approximations (solver tolerances and deviation search) are treated as explicit objects that can be monitored, stress-tested, and accounted for when interpreting incentive metrics.

4 Mechanism class: solver-backed neural allocations

We now define the mechanism class that we train and evaluate. The guiding design choice is to separate (i) a flexible, learned *scoring rule* from (ii) an *exact* combinatorial feasibility layer implemented by a winner-determination

solver. Economically, this makes the allocation rule interpretable as "maximize an objective over feasible allocations," while operationally it mirrors how platforms already implement combinatorial assignment problems: a learned model proposes priorities, and a solver enforces the hard constraints.

4.1 Score network (learned priority rule)

Fix parameters θ. Given a bid profile b, the score network outputs a real-valued score for each bidder–bundle pair,

$$s_\theta(b) \;=\; \big(s_{\theta,iS}(b)\big)_{i\in N,\; S\in K} \in \mathbb{R}^{n\times k}.$$

We interpret $s_{\theta,iS}(b)$ as the mechanism's "priority" for assigning bundle S to bidder i at the reported profile b. The network may use the full profile b, not only bidder i's own report b_i, so the scoring rule can capture competition effects (e.g., favoring allocations that create tighter markets or reduce fragmentation).

Two modeling remarks clarify what is, and is not, imposed here. First, we do *not* require scores to equal bids or values; the scorer is a learned transformation of reports into an objective that the solver will maximize. This is precisely the lever that permits revenue-oriented behavior beyond welfare maximization. Second, we do *not* impose monotonicity of allocations in bids, which is typically needed for classical payment characterizations. Instead, we will later regulate incentives by explicitly penalizing measured regret during training.

In implementations, the scorer can be instantiated in a number of standard ways. One convenient parameterization embeds each bidder's bid vector b_i into a latent representation (via an MLP), aggregates information across other bidders using a permutation-invariant layer (e.g., DeepSets-style pooling or attention with symmetric aggregation), and then decodes bundle-specific scores. Concretely, a template is

$$h_i \;=\; \phi_\theta\Big(b_i,\; \mathrm{Agg}\big(\{\psi_\theta(b_\ell)\}_{\ell\neq i}\big)\Big), \qquad s_{\theta,iS}(b) \;=\; \rho_\theta(h_i, S),$$

where Agg is symmetric in the multiset of other bidders' features. This structure is not required for the theory, but it is useful in practice: it hard-codes the symmetry that bidder labels carry no economic content, improving sample efficiency and reducing spurious asymmetries that can translate into brittle incentives.

4.2 Exact winner determination on scores

Given scores $s_\theta(b)$, the mechanism selects an allocation by solving a deterministic integer optimization problem over the feasibility set $\mathcal{X}$:

$$x_\theta(b) \;\in\; \arg\max_{x\in\mathcal{X}} \langle s_\theta(b), x\rangle \;=\; \arg\max_{x\in\mathcal{X}} \sum_{i\in N}\sum_{S\in K} s_{\theta,iS}(b)\, x_{iS}.$$

This maximization is exactly a winner-determination problem with "virtual values" given by the learned scores. Because $\mathcal{X}$ already encodes integrality and combinatorial feasibility, the output allocation is implementable by construction; there is no rounding step whose economic meaning would need to be justified.

A subtle but important point is that the argmax correspondence may not be single-valued when there are ties in the total score. For economic analysis (utilities, regret, and measurability of the induced outcome function), it is convenient to work with a well-defined selection $x_\theta(b)$. We therefore assume a deterministic tie-breaking rule that depends only on $(s_\theta(b), b)$ in a measurable way. Two standard approaches suffice. The first is *lexicographic* tie-breaking: among score-maximizing allocations, choose the allocation with the smallest index under a fixed ordering of (i, S)-pairs. The second is *infinitesimal perturbation*: add a fixed, vanishingly small priority vector $\varepsilon \cdot \pi$ (with distinct entries in π) to the score matrix before solving, which breaks ties without materially changing objectives. Either method yields a single-valued mapping $b \mapsto x_\theta(b)$ and rules out knife-edge discontinuities driven purely by indifference.

We emphasize that the solver is a *module* in the mechanism. From the platform's perspective, it is an auditable component: the allocation is the solution to a transparent integer program whose constraints are fixed and whose objective coefficients are the scores produced by the learned model. This modularity is central to the broader motivation. Many learning-based mechanisms differentiate through relaxations during training, but what is actually deployed must obey the hard allocation constraints. By defining the economic object directly in terms of the solver-backed integral allocation, we ensure that feasibility is not an empirical property but a logical one.

4.3 Payments with "IR-by-design" scaling

The payment rule is parameterized to guarantee ex-post individual rationality under truthful bidding, while still allowing the mechanism to learn nontrivial pricing. For each bidder i, we compute a scaling factor

$$\alpha_{\theta,i}(b) \in [0, 1],$$

and charge

$$p_{\theta,i}(b) \;=\; \alpha_{\theta,i}(b) \sum_{S \in K} x_{\theta,iS}(b)\, b_{iS}. \tag{7}$$

Because $x_{\theta,iS}(b)$ selects at most one bundle, the payment is simply $\alpha_{\theta,i}(b)\, b_{iS^\star}$ if bidder i receives $S^\star$, and 0 otherwise. This form has two practical advantages. First, it automatically respects a strong payment cap: no bidder pays more than her own reported bid for the allocated bundle. Second, it decouples allocation feasibility from payments: payments can be adjusted without affecting the constraint satisfaction of $x_\theta(b)$.

The scaling factor $\alpha_{\theta,i}(b)$ can itself be produced by a neural network (with a final sigmoid or clipping to enforce $[0, 1]$). Architecturally, we typically let $\alpha_{\theta,i}$ depend on features that summarize bidder i's competitive environment—for example, the gap between the best and second-best score-maximizing allocations involving i, or pooled statistics of others' bids. This allows the mechanism to learn pricing patterns such as charging more when competition is intense, while remaining within the IR envelope implied by (7).

It is important to be explicit about the limitation of this payment family. The rule (7) does not, by itself, ensure incentive compatibility: a bidder can manipulate both (i) whether she wins and (ii) the price conditional on winning through the dependence of $\alpha_{\theta,i}(b)$ on reports. In other words, we are trading the classical closed-form characterization of DSIC mechanisms for a design that hard-codes IR and feasibility while leaving incentive alignment to be encouraged (but not guaranteed) via regret minimization. The value of the approach is not that it eliminates strategic behavior in general, but that it yields a mechanism that is always implementable and safe from obvious participation failures, and whose remaining incentive issues can be measured and targeted directly.

4.4 Contextual inputs and symmetry (optional but useful)

Many applications require the mechanism to condition on side information beyond bids: item attributes, bidder segments, time-of-day effects, or reserve-price policies. We accommodate this by allowing the scorer and the payment scaler to depend on observed context c (common knowledge and non-strategic), writing $s_\theta(b, c)$ and $\alpha_{\theta,i}(b, c)$. Nothing in the allocation or payment definitions changes; context is simply an additional input to the learned components. This is useful in practice because it lets the mechanism adapt to predictable demand shifts without treating them as strategic signals embedded in b.

When using context, it becomes even more important to respect economic symmetries. At a minimum, bidder identities are arbitrary labels, so the mechanism should be permutation-equivariant in bidders: permuting bidders in the input should permute the scores and payments in the output. Analogously, if items are exchangeable within known categories, one may impose structured equivariance in how bundles are represented. These inductive biases do not alter the formal mechanism class, but they influence which θ are reachable under finite data and therefore shape the attainable revenue–regret frontier.

4.5 A note on DSIC special cases

Although our baseline class is designed for approximate incentive alignment, it nests a classical DSIC construction as a special case. If we restrict the

scorer to an affine-maximizer form, $s_{\theta,iS}(b) = w_i b_{iS} + \kappa_S$ with fixed weights and bundle boosts, and replace (7) with the corresponding VCG payment computed on these transformed bids, then the resulting solver-backed mechanism is DSIC and ex-post IR. We view this as a useful anchor: it shows that the solver-backed architecture is compatible with exact dominant-strategy truthfulness when one is willing to impose the relevant structure. Our main focus, however, is on the more flexible class above, which can depart from affine maximization and learn richer allocation heuristics, while using regret-based training to control the strategic distortions that this flexibility introduces.

5 Training as bilevel optimization

Having specified how a parameter vector θ maps reports b into an allocation $x_\theta(b)$ and payments $p_\theta(b)$, we now describe how we *choose* θ from data. The economic tension is familiar: a revenue-seeking platform would like to learn aggressive allocation and pricing rules, but doing so typically creates profitable misreports. In our framework we do not attempt to enforce dominant-strategy truthfulness by construction (except in the special cases noted earlier); instead, we train θ to trade off revenue against a direct, ex-post measure of strategic vulnerability.

5.1 Outer objective: revenue with regret control

We assume access to a distribution $\mathcal{D}$ over valuation profiles $v = (v_1, \ldots, v_n)$, either from a structural model, a simulator, or historical estimates. Training proceeds on i.i.d. samples $v \sim \mathcal{D}$, where we interpret the "truthful" report as $b = v$. For a fixed θ, define realized revenue and bidder utilities at a profile b as

$$\mathrm{rev}_\theta(b) \;=\; \sum_{i \in N} p_{\theta,i}(b), \qquad u_i(v_i; b) \;=\; \sum_{S \in K} x_{\theta,iS}(b)\, v_{iS} - p_{\theta,i}(b).$$

The key incentive metric is ex-post regret: for each i, holding other reports fixed at truth, we compare truthful utility to the best attainable utility from unilateral deviation. At a valuation profile v, the (true) ex-post regret is

$$\mathrm{rgt}_i(\theta; v) \;=\; \max_{b_i' \in \mathcal{B}_i} \; u_i\big(v_i; (b_i', v_{-i})\big) - u_i\big(v_i; (v_i, v_{-i})\big), \tag{8}$$

where $\mathcal{B}_i \subseteq \mathbb{R}_+^K$ is the admissible bid set (often $[0, V_{\max}]^K$, possibly with additional structure such as sparsity or budget caps). In training we combine revenue and regret in one of two equivalent ways:

$$\max_\theta \; \mathbb{E}_{v \sim \mathcal{D}}\Big[\mathrm{rev}_\theta(v) - \lambda \sum_{i \in N} \widehat{\mathrm{rgt}}_i(\theta; v)\Big] \tag{9}$$

for a penalty weight $\lambda > 0$, or

$$\max_{\theta} \ \mathbb{E}_{v\sim\mathcal{D}}\big[\mathrm{rev}_\theta(v)\big] \quad \text{s.t.} \quad \mathbb{E}_{v\sim\mathcal{D}}\big[\widehat{\mathrm{rgt}}_i(\theta; v)\big] \le \varepsilon \ \ \forall i \tag{10}$$

for a target regret level $\varepsilon \ge 0$. The penalty form (9) is convenient for stochastic gradient methods; the constrained form (10) makes the economic interpretation explicit ("we are willing to tolerate at most ε of ex-post deviation gains") and motivates primal–dual training procedures.

Two remarks matter for interpretation. First, $\widehat{\mathrm{rgt}}_i(\theta; v)$ is necessarily *estimated* by an inner optimization routine, so it is a lower bound on $\mathrm{rgt}_i(\theta; v)$ unless the inner problem is solved exactly. Second, because our mechanism is solver-backed and may be discontinuous in bids, gradients are not classical everywhere; training is therefore a numerical procedure whose outputs we evaluate with explicit deviation search, and whose approximation error we later track via δ (oracle suboptimality) and η (solver suboptimality).

5.2 Inner problem: deviation search as an "attacker"

For each sampled v and each bidder i, we require an approximate maximizer of $b_i' \mapsto u_i(v_i; (b_i', v_{-i}))$. We treat this as a best-response computation against the mechanism induced by θ, with all other bids fixed to truth. Formally, the inner problem is

$$b_i^\star(\theta; v) \ \in \ \arg\max_{b_i' \in \mathcal{B}_i} \ u_i\big(v_i; (b_i', v_{-i})\big). \tag{11}$$

In practice, we compute an approximate solution $\widehat{b}_i(\theta; v)$ using a deviation oracle. A standard choice is projected gradient ascent in bid space, with multiple random restarts and a projection operator $\Pi_{\mathcal{B}_i}$ to enforce feasibility:

$$b_i^{(t+1)} \ = \ \Pi_{\mathcal{B}_i}\Big(b_i^{(t)} + \gamma_t \nabla_{b_i} u_i\big(v_i; (b_i^{(t)}, v_{-i})\big)\Big),$$

where gradients are computed through the mechanism implementation using one of the strategies discussed below. Because the objective can be non-concave and non-smooth, we do not interpret this as "solving" a well-behaved optimization problem; rather, it is an adversarial search for profitable deviations, and its quality is summarized by a suboptimality gap δ in the sense that

$$u_i\big(v_i; (\widehat{b}_i(\theta; v), v_{-i})\big) \ \ge \ \max_{b_i' \in \mathcal{B}_i} u_i\big(v_i; (b_i', v_{-i})\big) \ - \ \delta$$

(typically in expectation over the oracle randomness and training samples). This δ is not merely a computational nuisance: it has a direct economic meaning, since a weak attacker can make a mechanism appear more incentive compatible than it is. We therefore report regret measured with the strongest deviation oracle we can afford computationally, and we later state performance bounds in terms of δ.

5.3 Three gradient strategies

The central computational difficulty is that both the allocation $x_\theta(b)$ and the deviation $\widehat{b}_i(\theta; v)$ depend on θ through argmax operators. We highlight three practical approaches, each with different biases and computational regimes.

(i) LP relaxation with (implicit) differentiation. A common approach is to replace the discrete winner-determination problem by a continuous relaxation during backpropagation. Let $\tilde{\mathcal{X}}$ be a polyhedral relaxation of $\mathcal{X}$ (e.g., the standard LP relaxation of the assignment constraints), and define

$$\tilde{x}_\theta(b) \;\in\; \arg\max_{x \in \tilde{\mathcal{X}}} \langle s_\theta(b), x \rangle. \tag{12}$$

To obtain useful gradients, we typically add a strongly convex regularizer, such as $-\tau H(x)$ where H is an entropy-like term (or a quadratic prox), yielding a unique and smooth solution map $s \mapsto \tilde{x}(s)$ for $\tau > 0$. One can then differentiate $\tilde{x}_\theta(b)$ with respect to scores $s_\theta(b)$ via the KKT conditions and implicit differentiation, and backpropagate further through $s_\theta(\cdot)$ and $\alpha_\theta(\cdot)$. This route tends to produce low-variance gradients and stable optimization when the relaxation is tight and the regularization is well tuned.

The drawback is conceptual: the forward pass used at deployment is integral and solver-backed, while the backward pass is driven by $\tilde{x}_\theta$. When the integrality gap is material, training may optimize the relaxed objective rather than the deployed one. We treat this as a training–deployment mismatch and later relate performance gaps to $\|x_\theta(b) - \tilde{x}_\theta(b)\|_1$ and payment sensitivity. Practically, we have found LP-based differentiation most appropriate when (a) instances are large enough that exact differentiation through an IP solver is infeasible, and (b) the relaxation is empirically tight on the distribution of interest.

(ii) Straight-through estimators (STE) with exact forward allocation. A second approach keeps the *exact* integral solver in the forward pass, but uses a surrogate gradient in the backward pass. Concretely, we compute $x_\theta(b)$ by the true integer argmax over $\mathcal{X}$, but when backpropagating we pretend the mapping $s \mapsto x$ is differentiable and substitute $\partial x/\partial s$ with a proxy (often the identity map, or the Jacobian of a smoothed softmax/LP layer evaluated at the same scores). This is the straight-through idea:

$$\text{forward: } x \leftarrow \arg\max_{x \in \mathcal{X}} \langle s, x \rangle, \qquad \text{backward: } \frac{\partial \mathcal{L}}{\partial s} \approx \frac{\partial \mathcal{L}}{\partial x} \cdot \widehat{J}(s).$$

The appeal is that the learned parameters see gradients that "respect" the discrete allocation actually used to compute revenue and utilities, so the optimization is not pulled toward artifacts of a loose relaxation. The cost is bias: the gradients are not gradients of any true objective in general, and

convergence guarantees are weak. Nevertheless, STE often performs well when the integer program is fast to solve (so we can afford many forward calls, including within the deviation oracle), and when relaxations are too loose to be informative.

From an economic standpoint, STE is best viewed as a heuristic that searches parameter space for mechanisms with good empirical revenue–regret tradeoffs under a given evaluation oracle. This perspective aligns with our later emphasis on *auditable* properties (exact feasibility and the payment cap) and *measured* incentive performance rather than purely analytic optimality.

(iii) Primal–dual and cutting-plane surrogates for regret constraints. A third route is to train the mechanism using a constraint-generation view of regret minimization, which reduces reliance on differentiating through the inner maximization (11). Consider the constrained program (10) and form a Lagrangian with multipliers $\mu_i \geq 0$:

$$\mathcal{L}(\theta, \mu) \;=\; \mathbb{E}_v\big[\mathrm{rev}_\theta(v)\big] \;-\; \sum_{i \in N} \mu_i \Big(\mathbb{E}_v\big[\widehat{\mathrm{rgt}}_i(\theta; v)\big] - \varepsilon \Big).$$

Training alternates between (a) approximately maximizing $\mathcal{L}$ over θ (using whichever gradient proxy is available for rev_θ and for the regret term evaluated at the currently found deviations), and (b) updating μ by (stochastic) subgradient ascent on constraint violations:

$$\mu_i \leftarrow \big[\mu_i + \beta\big(\widehat{\mathrm{rgt}}_i - \varepsilon\big)\big]_+.$$

A closely related cutting-plane interpretation maintains, for each bidder, a growing set of "witness" deviations $\mathcal{W}_i$ found by the oracle. Instead of the max in (8), we penalize the largest utility gain among the currently known witnesses, periodically refreshing $\mathcal{W}_i$ by attacking the latest θ. This method is attractive when the inner maximization is expensive or unstable: rather than differentiating through the argmax, we treat it as a separation oracle that identifies violated incentive constraints. Economically, it mirrors how one would audit a mechanism: search for a profitable deviation, add it to the test suite, and retrain to remove the vulnerability.

5.4 When each approach is appropriate

The choice among these strategies is ultimately governed by instance scale and by what we want to be "exact." LP-based differentiation tends to be the method of choice when we need smooth, low-variance gradients and can tolerate some mismatch between relaxed and deployed allocations; it is also well suited to large markets where only approximate winner determination is feasible during training. STE is most compelling when we insist that the forward pass reflect the true deployed mechanism and have access to a fast exact

(or near-exact) solver; it typically pairs well with strong deviation search, since the oracle is attacking precisely the mechanism that will be evaluated. Primal–dual and cutting-plane approaches are useful when we want explicit control of regret constraints and are willing to treat deviations as adversarial examples that are iteratively discovered rather than fully optimized at every step.

In all cases, we emphasize the same discipline: training is a numerical procedure with approximations, so we report both revenue and *measured* regret under a specified oracle, and we interpret results through the lens of approximation gaps. The next section formalizes which properties are exact by construction (feasibility and truthful IR) and how solver and oracle errors (η, δ) propagate into bounds on the true regret of the learned mechanism.

6 Theory: exact constraints and approximation-aware incentive guarantees

Our mechanism class is intentionally modular: a learned scorer proposes *what* the platform would like to do, while a winner-determination solver enforces *what the platform is allowed to do*. This separation is economically useful. It lets us make two statements that are fully distribution-free and *ex post* (they hold for every realized bid profile), while treating incentive performance as an empirical object that we control through regret minimization and quantify through approximation bounds.

6.1 Feasibility and implementability are enforced by the solver

In combinatorial auctions, the first-order operational requirement is that the mechanism output a valid assignment of items and bundles: no item can be allocated twice, and no bidder can receive two bundles. Many differentiable approaches relax this requirement during training and only enforce it approximately at deployment, which can create additional complications (randomization to restore feasibility, decomposition of fractional allocations, etc.). In contrast, we treat feasibility as a hard constraint built into the allocation rule.

Formally, recall that the feasible set $\mathcal{X}$ consists of binary allocation tensors $x = (x_{iS})_{i \in N, S \in K}$ satisfying the standard item non-overlap constraints and the at-most-one-bundle-per-bidder constraints. Given bids b, the score network computes $s_\theta(b)$, and the platform computes

$$x_\theta(b) \;\in\; \arg\max_{x \in \mathcal{X}} \langle s_\theta(b), x \rangle.$$

Because the maximization is taken *over* $\mathcal{X}$, every returned allocation is integral and combinatorially feasible by construction. The only subtlety is

tie-breaking: if the score objective admits multiple maximizers, we must select one in a deterministic, measurable way (e.g., lexicographic tie-breaking on the vector x, or by adding an infinitesimal perturbation to scores that is fixed ex ante). Once we do so, $x_\theta(\cdot)$ is a single-valued rule.

Economically, we interpret this as an implementability guarantee. The platform can audit the output allocation without reference to training data or distributional assumptions. This matters in practice because feasibility failures are not merely a theoretical nuisance: they correspond to physical infeasibilities (allocating the same item twice) and to contractual ambiguities (a bidder receiving multiple bundles when only one is permitted).

6.2 Ex-post individual rationality via a payment cap

The second property we build in is a conservative payment structure that ensures truthful bidders never face negative utility. The payment charged to bidder i at bid profile b is

$$p_{\theta,i}(b) \;=\; \alpha_{\theta,i}(b) \sum_{S \in K} x_{\theta,iS}(b)\, b_{iS}, \qquad \alpha_{\theta,i}(b) \in [0,1].$$

This rule is intentionally simple: we charge at most the bidder's own reported value for whatever bundle she receives, scaled by $\alpha_{\theta,i}(b)$. The learned component $\alpha_{\theta,i}(b)$ can be interpreted as a "discount factor" that trades off revenue against robustness (high α extracts more surplus but can raise incentives to shade or misreport).

To see why this delivers ex-post IR under truthful bidding, fix bidder i with true values v_i, fix any profile of other bids b_{-i}, and suppose bidder i reports $b_i = v_i$. Then her realized utility is

$$\begin{aligned} u_i\big(v_i; (v_i, b_{-i})\big) &= \sum_{S \in K} x_{\theta,iS}(v_i, b_{-i})\, v_{iS} \;-\; p_{\theta,i}(v_i, b_{-i}) \\ &= \sum_{S \in K} x_{\theta,iS}(v_i, b_{-i})\, v_{iS} \;-\; \alpha_{\theta,i}(v_i, b_{-i}) \sum_{S \in K} x_{\theta,iS}(v_i, b_{-i})\, v_{iS} \\ &= \big(1 - \alpha_{\theta,i}(v_i, b_{-i})\big) \sum_{S \in K} x_{\theta,iS}(v_i, b_{-i})\, v_{iS} \;\geq\; 0, \end{aligned}$$

since $\alpha_{\theta,i} \in [0,1]$ and values are nonnegative. This is an *ex-post* statement: it holds for every realized (v_i, b_{-i}), not merely in expectation over a training distribution. The guarantee is also auditable at deployment time: it relies only on the inequality $\alpha \in [0,1]$, which is easy to enforce by parameterization (e.g., a sigmoid output).

We emphasize what this guarantee does *not* do. It does not imply incentive compatibility: a bidder may still profitably deviate (indeed, training is designed to reduce such opportunities). Nor does it guarantee IR under arbitrary misreports; if a bidder inflates her bid, the payment cap is relative

to her own report, so she can always "talk herself into" an unaffordable payment. The point is narrower and operational: under truthful reporting, the mechanism never charges more than realized value for the allocated bundle.

6.3 Regret stability under solver and deviation-oracle approximation

We next formalize how two computational approximations affect measured incentive performance.

Solver approximation. Even when we conceptually define $x_\theta(b)$ via an exact argmax, deployments often allow a solver optimality tolerance, or use heuristics that return a near-optimal solution. To model this, let $x^\star(b)$ denote an exact maximizer of $\langle s_\theta(b), x\rangle$ over $\mathcal{X}$, and suppose the solver returns $x_\theta(b)$ satisfying

$$\langle s_\theta(b), x^\star(b)\rangle - \langle s_\theta(b), x_\theta(b)\rangle \;\le\; \eta \qquad \text{for all } b. \tag{13}$$

Feasibility is unchanged: as long as the solver optimizes over $\mathcal{X}$, the returned solution remains integral and feasible, regardless of η. What changes is welfare (in score space), and through it, utilities and regret.

Deviation-oracle approximation. Regret is defined by a best-response computation. In training and evaluation we approximate it with an "attacker" that searches for profitable deviations. Let $\widehat{b}_i(\theta; v)$ be the deviation found by the oracle for bidder i at profile v. We summarize the oracle quality by δ in the standard additive sense:

$$u_i\big(v_i; (\widehat{b}_i(\theta; v), v_{-i})\big) \;\ge\; \max_{b_i' \in \mathcal{B}_i} u_i\big(v_i; (b_i', v_{-i})\big) \;-\; \delta. \tag{14}$$

The measured regret uses $\widehat{b}_i$, while the true regret uses the exact maximizer.

A Lipschitz bridge from allocation error to utility error. To connect η to incentives we need a regularity assumption that bounds how sensitive payments (and hence utilities) are to changes in the allocation. A convenient sufficient condition is that, for each i and fixed b, the payment functional is L-Lipschitz in the allocation argument in ℓ_1:

$$\big|p_i(x, b) - p_i(x', b)\big| \;\le\; L\|x - x'\|_1. \tag{15}$$

Because values are bounded by $V_{\max}$ and allocations are binary with limited support (each bidder receives at most one bundle), utility is also Lipschitz in x up to constants that depend on $V_{\max}$ and the size of the market. Intuitively, a solver error can only change a bidder's payoff if it changes whether she wins and which bundle she wins; with bounded values and Lipschitz payments, the corresponding payoff impact is bounded.

Regret underestimation bound. Combining (13)–(15) yields a stability statement of the following form: the regret we measure with an approximate oracle and an approximate solver upper-bounds the true regret up to additive terms that scale with δ and η. Concretely, letting $\mathrm{rgt}_{\mathrm{true}}(\theta)$ denote expected regret with exact best responses and exact score-maximizing allocations, and $\mathrm{rgt}_{\mathrm{measured}}(\theta)$ denote the corresponding quantity computed with the δ-oracle and the η-optimal solver, we obtain

$$\mathrm{rgt}_{\mathrm{true}}(\theta) \;\le\; \mathrm{rgt}_{\mathrm{measured}}(\theta) \;+\; \delta \;+\; O(L\eta), \tag{16}$$

where the hidden constant depends on how score suboptimality translates into ℓ_1 allocation deviations on the encountered instances (in particular, on the normalization and range of scores and on the effective size of $\mathcal{X}$). The economic interpretation is direct: a weaker attacker (δ large) mechanically understates manipulability, and a looser solver (η large) introduces additional noise into utilities that can create or destroy profitable deviations. Importantly, neither approximation breaks the *hard* properties above (feasibility and truthful IR), but both can materially affect incentive performance.

6.4 Training on relaxations and deploying integral allocations

Finally, we address a mismatch that arises when we differentiate through a relaxation. During training, a common choice is to replace the integer program over $\mathcal{X}$ by a continuous relaxation $\tilde{\mathcal{X}} \supseteq \mathcal{X}$, producing a fractional optimizer $\tilde{x}_\theta(b)$. The hope is computational: $\tilde{x}_\theta(b)$ depends smoothly on scores (often after adding a regularizer), giving low-variance gradients. Deployment, however, uses the integral $x_\theta(b)$ from the true solver.

This introduces a conceptually simple but practically important question: how far can performance under the relaxation deviate from performance under the deployed integral rule? A useful way to answer it is to treat revenue and utilities as functions of (x, b) and to bound their variation in x. Under Lipschitz payments (and bounded values), we can control the gap between relaxed and integral outcomes by the distance $\|x_\theta(b) - \tilde{x}_\theta(b)\|_1$. For example, for revenue one can write

$$\big|\mathrm{rev}(x_\theta(b), b) - \mathrm{rev}(\tilde{x}_\theta(b), b)\big| \;\le\; \sum_{i \in N} \big|p_i(x_\theta(b), b) - p_i(\tilde{x}_\theta(b), b)\big| \;\le\; nL\|x_\theta(b) - \tilde{x}_\theta(b)\|_1,$$

and therefore, taking expectations over $v \sim \mathcal{D}$ (with $b = v$ in the truthful evaluation),

$$\Big|\mathbb{E}\big[\mathrm{rev}(x_\theta(v), v)\big] - \mathbb{E}\big[\mathrm{rev}(\tilde{x}_\theta(v), v)\big]\Big| \;\le\; O\big(L \cdot \mathbb{E}\big[\|x_\theta(v) - \tilde{x}_\theta(v)\|_1\big]\big), \tag{17}$$

with an analogous bound for utilities and regret.

Equation (17) makes the key point: the relevant object is not an abstract worst-case integrality gap, but the *instance-weighted* mismatch between the relaxed solution used for gradients and the integral solution used for actual allocations. When the relaxation is tight on the distribution of interest, this mismatch is small, and training by relaxed differentiation can be well aligned with deployment. When the mismatch is large, gradients can systematically optimize a surrogate that the deployed mechanism does not implement, potentially yielding disappointing out-of-sample revenue or brittle incentives.

This discussion also clarifies why we treat feasibility and IR as "by design" properties, while treating incentive compatibility as an empirical target. Feasibility and truthful IR hold exactly for the deployed mechanism regardless of how we compute gradients. Regret, by contrast, depends on how well training attacks reflect true best responses and on how closely the training-time surrogate reflects the deployment-time allocation. The purpose of our experiments is therefore not only to compare revenue levels, but to stress-test regret under stronger deviation search and to quantify how solver tolerances and gradient choices move the mechanism along the revenue–robustness frontier.

7 Experiments: synthetic combinatorial auctions and scaling behavior

Our experiments serve three purposes. First, we quantify the revenue–robustness tradeoff induced by regret-regularized training in small combinatorial auctions where the feasible bundle set is rich and strategic incentives are nontrivial. Second, we benchmark against recent learned allocation rules (e.g., CAFormer/CANet-style architectures) and simple operational heuristics that practitioners often deploy when full mechanism design is infeasible. Third, we stress-test the incentive claims by evaluating regret under increasingly strong deviation search, and we document the computational footprint of solver-backed deployment (solve times, optimality certificates, and sensitivity to solver tolerances and gradient choices).

7.1 Environments and synthetic valuation models

We consider synthetic combinatorial auctions with $(n, m) \in \{(2, 2), (2, 3), (2, 5)\}$. For these small markets we take the allowable bundle set to be the full non-empty powerset, $K = 2^M \setminus \{\emptyset\}$, so that bidders may value any combination of items. This choice deliberately creates complementarity patterns that cannot be reduced to item-wise bidding without loss. Values are bounded and normalized to $v_{iS} \in [0, V_{\max}]$, which allows direct comparison across environments and aligns with our approximation bounds.

To generate valuation profiles, we use a mixture of structured and un-

structured models. Concretely, for each bidder we first sample base item values $(a_{ij})_{j \in M}$ i.i.d. from a bounded distribution (e.g., uniform on $[0, 1]$), and then construct bundle values as

$$v_{iS} \;=\; \min\Big\{V_{\max},\; \sum_{j \in S} a_{ij} \;+\; \sum_{\{j,\ell\} \subseteq S} \gamma_{i,j\ell}\Big\},$$

where $\gamma_{i,j\ell}$ are pairwise synergy terms (possibly negative) drawn from a mean-zero bounded distribution and truncated to preserve nonnegativity. This produces a controlled spectrum from nearly additive preferences (small synergies) to strongly complementary preferences (large positive synergies), while maintaining bounded support. We also include a *single-minded* component in which, with some probability, bidder i draws a target bundle T_i and sets v_{iT_i} high relative to other bundles; this is a useful stress case because profitable deviations can hinge on winning a specific bundle rather than marginal shading.

Throughout, we evaluate *truthful* performance at bid profiles $b = v$ (revenue, allocations, and individual rationality), and we evaluate strategic robustness by computing ex-post regret with respect to misreports b_i' drawn from a rich continuous action space.

7.2 Mechanism training and evaluation protocol

We train the solver-backed neural mechanism by minimizing a standard revenue–regret objective on i.i.d. valuation samples,

$$\min_{\theta} \;\mathbb{E}_v\Big[-\mathrm{rev}(v) \;+\; \lambda \sum_{i \in N} \widehat{\mathrm{rgt}}_i(\theta; v)\Big],$$

where $\widehat{\mathrm{rgt}}_i$ is computed by an inner deviation oracle (attacker) that searches for a profitable misreport against truthful opponents. We treat λ as a tuning parameter that traces a Pareto frontier: larger λ produces mechanisms with lower measured regret but typically lower revenue.

A central methodological point is that the deviation oracle is not only a training tool but also an evaluation tool. Accordingly, we report two regret numbers for each learned θ: (i) an *in-training* regret using the same oracle class used during learning, and (ii) an *out-of-training stress-test* regret using a strictly stronger oracle (more restarts, larger iteration budget, and hybrid local–global search). This mirrors standard adversarial evaluation: a mechanism is only as robust as the strongest deviations it fails to prevent.

7.3 Baselines: learned and heuristic comparators

We compare against two classes of baselines.

Learned allocation baselines. We include architectures in the CAFormer/CANet family that directly output (or approximate) allocations from bids using permutation-equivariant neural components. Because many such methods rely on differentiable relaxations or sampling-based rounding, we evaluate them under their recommended training and deployment pipelines. Where the baseline outputs fractional allocations, we apply the baseline's own rounding/post-processing rule to obtain a deterministic feasible allocation for a fair revenue and regret comparison. We emphasize that our main comparison axis is *deployed* behavior: feasibility, revenue, and regret computed on the final integral allocation.

Heuristic mechanisms. We implement simple operational heuristics that are commonly used when full combinatorial optimization or principled payments are not available: (i) greedy winner determination on reported bundle bids (sorting by bid or bid-per-item, then accepting non-overlapping bundles), coupled with pay-as-bid payments; (ii) item-wise posted pricing learned from data (allocating items or small bundles independently), which is not expressive for complementarities but is fast; and (iii) a fixed-α pay-your-bid scaling rule $p_i = \alpha \sum_S x_{iS} b_{iS}$ with α chosen by validation to illustrate the limits of non-adaptive pricing. These heuristics provide a useful "floor" for both revenue and robustness: they are easy to deploy, but their incentive properties are typically poor and their allocations can be far from score-optimal when complementarities matter.

7.4 Metrics and solver instrumentation

We report (a) expected truthful revenue $\mathbb{E}[\mathrm{rev}(v)]$; (b) expected ex-post regret $\mathbb{E}[\max_{b_i'} u_i(v_i; (b_i', v_{-i})) - u_i(v_i; (v_i, v_{-i}))]$, estimated by the specified deviation oracle; and (c) feasibility and truthful IR violation rates. In our solver-backed class, feasibility violations are identically zero by construction, and truthful IR violations are identically zero given the enforced $\alpha_{\theta,i}(b) \in [0, 1]$; we still log these statistics as sanity checks and to make comparisons to baselines transparent.

Because our allocation rule calls a combinatorial optimizer, we also report computational statistics that matter operationally: median and tail solve times per instance, the distribution of optimality gaps when a tolerance is used, and the frequency of tie cases under the chosen deterministic tie-breaker. These logs are not ancillary: they reveal whether a mechanism that is conceptually attractive is actually deployable at the transaction latencies relevant to the application domain.

7.5 Results in small markets: revenue–regret frontiers

Across the 2×2, 2×3, and 2×5 environments, we consistently observe a smooth revenue–regret frontier as λ varies. Economically, this is the expected pattern: pushing the mechanism toward higher revenue typically creates sharper allocation discontinuities in bid space and stronger incentives to manipulate; increasing the regret weight dampens these incentives, often by learning more conservative payment scaling $\alpha_{\theta,i}(b)$ and by adjusting scores to reduce the gains from small misreports.

Relative to CAFormer/CANet-style baselines, the solver-backed approach tends to dominate in regimes where complementarities are important. The underlying reason is straightforward: when bundle interactions matter, errors in feasibility or rounding can change who wins which bundle, and regret is highly sensitive to such winner changes. By separating the learned scoring from the exact feasibility enforcement, we can focus learning capacity on ranking allocations while delegating integrality to the solver. Relative to greedy heuristics, the gains are larger: greedy allocation frequently sacrifices high-value complementary bundles, and pay-as-bid pricing combined with greedy selection tends to yield high regret under even moderate deviation search.

7.6 Regret stress tests: stronger deviation oracles

We next evaluate robustness under a strictly stronger deviation oracle than the one used in training. Practically, we increase the attacker budget (iterations and restarts) and augment gradient-based search with coordinate perturbations and random search over bid vectors to escape local maxima of the deviation objective. Two qualitative findings are robust.

First, regret estimates increase for all methods under stronger attacks, which is expected: regret is a supremum over deviations, so better search finds more profitable misreports. Second, the *gap* between in-training regret and stress-test regret is materially smaller for mechanisms trained with higher λ, consistent with the interpretation that regret regularization makes the utility landscape less exploitable. This exercise operationalizes the δ-dependence discussed earlier: the stronger oracle is simply a smaller-δ approximation to best response, and mechanisms that appear "IC" under weak attacks often do not remain so under stronger ones.

Importantly, the stress test also clarifies a practical governance point: a platform that reports only a single regret number without documenting the attacker strength is not providing an incentive guarantee in any meaningful sense. In our reporting, the attacker configuration is treated as part of the experimental specification, and we view regret under multiple attacker strengths as the analogue of robustness curves in adversarial machine learning.

7.7 Scaling studies and runtime behavior

To probe scalability beyond the smallest instances, we conduct scaling studies in which m increases and the bundle set K is either truncated (e.g., all singletons and pairs) or sampled (a fixed budget of candidate bundles per bidder) to mimic realistic bidding languages. We evaluate how (i) winner-determination time, (ii) end-to-end mechanism latency, and (iii) achieved revenue and regret change with problem size.

The computational pattern is the expected one for solver-backed mechanisms: solve time grows with the combinatorial complexity of $\mathcal{X}$ and with the richness of K, but remains manageable when K reflects realistic bid submission constraints. Moreover, we find that warm-starting the solver with the previous solution (or with a greedy feasible allocation) materially improves median latency and reduces tail events in which the solver spends time proving optimality. These observations motivate deployment practices that treat the solver not as a black box but as a monitored subsystem with performance knobs and logging (a theme we return to in the next section).

7.8 Ablations: solver tolerance and gradient method

Finally, we ablate two design choices that directly connect computation to incentives.

Solver optimality tolerance. We vary the solver tolerance to induce different η-levels in the score objective. As tolerance loosens, we observe the anticipated tradeoff: solve times decrease, but both revenue and regret become noisier and, on average, regret increases. This is consistent with the economic logic that suboptimal allocations can create new profitable deviations by changing pivotal winner/loser events. The key practical implication is that "faster" is not monotone-improving once incentives matter; solver tolerances should be treated as mechanism parameters that belong on the same frontier as λ.

Gradient estimation through the allocation module. We compare training with (i) differentiation through a continuous relaxation of winner determination (with a fractional optimizer used only for gradients) and (ii) straight-through estimators that treat the solver output as if it were differentiable. The main difference is not feasibility—deployment is integral in both cases—but *alignment*: relaxation-based gradients can be lower variance yet can optimize a surrogate that differs from the deployed rule, while straight-through gradients can be noisy but directly target the deployed decision. Empirically, we find that tighter relaxations improve this alignment and reduce the need for heavy regret regularization to achieve a given robustness level.

Taken together, these experiments emphasize the paper's core message: exact feasibility and truthful IR can be treated as non-negotiable operational constraints, while incentive compatibility is an empirical robustness target whose credibility depends on (i) the strength of deviation search used in evaluation and (ii) the computational approximations used in winner determination and training. The next section turns to how these properties can be audited and documented at deployment time.

8 Auditing and deployment: from a learned mechanism to an accountable system

A learned mechanism is not only an economic object but also an operational pipeline: bids arrive, scores are computed, a discrete winner-determination problem is solved, and payments are charged. This section describes how we make that pipeline auditable in a way that is meaningful both economically (does the deployed rule match the intended rule?) and institutionally (can a platform credibly justify outcomes to participants, regulators, and internal risk teams?). Our guiding premise is simple: because the allocation is computed by an optimizer over an explicit feasibility set, the mechanism naturally produces artifacts—objective values, certificates, and deterministic decisions—that can be logged and later re-verified.

8.1 Reproducibility of allocations as a first-class design requirement

Reproducibility is the minimal standard for accountability: for a fixed bid profile b, a bidder (or auditor) should be able to determine whether the deployed allocation $x_\theta(b)$ could have been produced by the announced mechanism. Achieving this requires more than saving θ. In practice we must pin down *all* degrees of freedom that can affect the solver-backed map $b \mapsto x_\theta(b)$.

First, we fix deterministic tie-breaking. Even when the objective $\langle s_\theta(b), x\rangle$ is integral and the feasible set $\mathcal{X}$ is discrete, multiple maximizers can exist. We therefore implement a lexicographic tie-breaker (e.g., prioritize lower bidder index, then smaller bundle index) by adding an infinitesimal but deterministic perturbation to the score objective or by using the solver's built-in objective priority features. The key is that tie-breaking is part of the mechanism specification, not an implementation accident.

Second, we version all mechanism components. A reproducible deployment record includes (i) the score network identifier (architecture and weights hash), (ii) the precise bundle universe K and any preprocessing applied to bids, (iii) the optimizer model (constraints defining $\mathcal{X}$, objective scaling, and any additional business rules), and (iv) the solver build and parameter configuration. In our experience, solver parameters that are innocuous

for pure optimization (e.g., presolve aggressiveness, parallelism, or numerical tolerances) can matter for determinism and should be treated as controlled variables.

Third, we log the *complete* instance needed to replay the decision: the bid profile b, the score matrix $s_\theta(b)$, the chosen allocation $x_\theta(b)$, the payment scalars $\alpha_{\theta,i}(b)$, and the realized payments $p_{\theta,i}(b)$. Logging $s_\theta(b)$ is especially valuable because it separates questions about the learned mapping (did the scorer produce the announced scores?) from questions about the combinatorial optimization (did the solver maximize the announced score objective over $\mathcal{X}$?). This division of responsibility mirrors how real organizations allocate accountability across teams.

8.2 Logging solver certificates: optimality, gaps, and feasibility proofs

The distinctive advantage of solver-backed deployment is that the winner determination step can emit *verifiable evidence* about its own correctness. When the solver is run to optimality, we obtain a proof of optimality in the standard sense: a feasible integer solution $x_\theta(b)$ and a matching bound showing no better solution exists. When the solver is run with tolerance, we obtain a quantitative certificate of suboptimality.

Concretely, for each auction instance we log:

- the incumbent objective value $\langle s_\theta(b), x_\theta(b) \rangle$;
- the best bound on the optimum reported by the solver (e.g., the branch-and-bound dual bound);
- the resulting optimality gap, which we interpret as the realized η for that instance;
- runtime and node counts (useful for diagnosing tail latency and for identifying atypical instances);
- a feasibility report (e.g., confirmation that the returned $x_\theta(b)$ satisfies non-overlap and at-most-one-bundle constraints).

These logs allow an auditor to answer two economically relevant questions. The first is *implementability*: did the platform allocate a feasible set of bundles? This is immediate from the recorded $x_\theta(b)$ and the explicit constraints defining $\mathcal{X}$. The second is *fidelity to the scoring rule*: did the platform approximately maximize the announced objective? This is quantified by the recorded bound and gap, and it converts what would otherwise be a qualitative engineering claim into a numerical mechanism parameter.

Two practical refinements are worth emphasizing. (i) We store the solver model in a canonical form (including objective scaling and any big-M constants) to avoid the situation where a replay run differs because of innocuous-looking reformulations. (ii) We monitor "certificate health" over time: if the distribution of optimality gaps or runtimes drifts, this is an early signal of distribution shift in bids, changes in K, or numerical instability in scores, all of which are relevant for the incentive guarantees.

8.3 Transparency of the score function: what can be disclosed and what should be testable

A learned scorer $s_\theta(\cdot)$ is typically too complex to be communicated as a closed-form rule, yet some degree of transparency is essential for trust. We separate transparency into three layers: *specification transparency*, *behavioral transparency*, and *verifiable transparency*.

Specification transparency means documenting what the scorer depends on and what it does *not* depend on. In our setting, the scorer is a function of reported bids b (and fixed public objects such as K), not bidder identities beyond permutation-equivariant structure, and not external covariates. This matters for governance: if the platform claims that allocations are determined solely by submitted bids and feasibility, then the code path must reflect that claim.

Behavioral transparency means providing accessible explanations of why a bidder won or lost, even if the underlying function is complex. Because the allocation maximizes $\langle s_\theta(b), x\rangle$, a natural explanation is comparative: bidder i lost bundle S because the chosen allocation had higher total score, often due to conflicts on items $j \in S$. We can therefore generate a concise "blocking set" explanation by reporting which allocated bundles overlap with S and their associated scores. This is not a full incentive proof, but it is the kind of local reason that participants can understand and contest if something appears inconsistent.

Verifiable transparency is the strongest notion: an auditor can compute $s_\theta(b)$ for any b and check that deployment used those scores. One way to implement this without fully open-sourcing proprietary models is to provide a signed scoring service: for any submitted b, the service returns $s_\theta(b)$ with a cryptographic signature and a model hash, so that later the platform cannot claim a different scoring function was used. Combined with solver certificates, this creates an end-to-end chain: bids $\rightarrow$ signed scores $\rightarrow$ certified maximization $\rightarrow$ logged allocation and payments.

8.4 Auditing payments and individual rationality

Payment auditing is conceptually simpler than allocation auditing in our class because payments are computed by a direct formula once $x_\theta(b)$ is fixed.

For each bidder i, the logged tuple $\big(\alpha_{\theta,i}(b), \sum_S x_{\theta,iS}(b) b_{iS}, p_{\theta,i}(b)\big)$ is sufficient to verify that

$$p_{\theta,i}(b) = \alpha_{\theta,i}(b) \sum_{S \in K} x_{\theta,iS}(b) b_{iS}, \qquad \alpha_{\theta,i}(b) \in [0,1].$$

This check is operationally important: it allows a platform to demonstrate that it did not charge an amount exceeding the bidder's own reported willingness to pay for the allocated bundle. When bids are truthful (or when governance requires interpreting bids as binding commitments), this is exactly the type of property that risk and compliance teams look for.

We emphasize a limitation that is also a governance choice. Individual rationality "by design" is with respect to a bidder's *report* and the realized allocation. It does not ensure that bidders will *want* to participate under strategic misreports, nor does it prevent participants from harming themselves through poorly chosen bids. In applications where bidders are not sophisticated, additional guardrails (bid caps, bid validation, or user-interface constraints) may be warranted and should be viewed as part of the mechanism, not mere product design.

8.5 Randomized extensions (optional): how to preserve auditability

Although our deployed rule is deterministic, there are reasons a platform might consider randomized extensions: randomized tie-breaking to mitigate perceptions of unfairness, randomized smoothing to reduce sharp discontinuities in allocations, or explicit lotteries to approximate constrained-optimal mechanisms when deterministic rules perform poorly. Randomization, however, can easily destroy auditability if it is implemented informally.

If randomness is used, we recommend treating it as a public input to the mechanism. Operationally, we log the random seed and the pseudorandom generator specification, and we generate the seed via a commit–reveal protocol (e.g., commit to a hash of the seed before bids are opened, reveal after allocation). This prevents ex post manipulation of randomness to favor particular bidders while preserving replayability. For randomized tie-breaking, the seed simply selects among score-maximizing allocations; for more substantive randomization, the seed determines the sampled allocation from a declared distribution.

Randomization also changes what must be audited. Instead of verifying a single decision $x_\theta(b)$ as the argmax of a score objective, the auditor must verify that $x_\theta(b)$ is a valid draw from the declared distribution conditional on b. This is feasible when the distribution has a tractable sampling procedure with logged randomness, but it should be acknowledged as a nontrivial extension rather than an implementation detail.

8.6 Governance constraints: embedding policy into $\mathcal{X}$ and into reporting

Platforms rarely run auctions in a policy vacuum. Common constraints include eligibility rules, allocation caps, exclusion lists, geographic restrictions, and, in some domains, fairness or non-discrimination requirements. A practical advantage of solver-backed design is that many such constraints can be expressed directly as linear constraints and added to the feasible set, replacing $\mathcal{X}$ by a constrained $\mathcal{X}'$. When governance rules are encoded this way, compliance becomes auditable in exactly the same sense as item non-overlap: the allocation itself is a certificate of adherence.

Not all governance objectives fit cleanly into $\mathcal{X}$. Some are about *information* rather than feasibility: privacy constraints on what is logged, disclosure requirements about how outcomes are determined, or audit access controls. Here we recommend a layered logging policy: store full bid and score records in an access-controlled vault with retention limits, while publishing a redacted public transcript (allocation, payments, and aggregated solver statistics) sufficient to establish procedural fairness without exposing sensitive bid data. For third-party audits, secure enclaves or privacy-preserving attestations can allow verification of solver certificates and payment identities without disclosing bidder-level bids.

Finally, governance must confront the fact that approximate incentive guarantees are only as credible as their monitoring regime. We therefore advocate a deployment-time "incentive dashboard": periodic regret stress tests on recent bid distributions (using held-out attacker budgets), monitoring of solver gaps η, and alerting when these quantities drift. The goal is not to claim static incentive compatibility, but to institutionalize a process by which the platform can detect when previously learned parameters θ no longer provide the desired robustness.

Taken together, these auditing practices turn a solver-backed learned mechanism into a system that can be inspected, replayed, and governed. They also clarify what remains open: scaling these assurances to richer bidding languages, producing stronger (and cheaper) incentive certificates, and updating mechanisms safely under distribution shift. We turn to these next-step questions in the conclusion.

9 Conclusion and next steps

We have argued for a particular division of labor in learned mechanism design: use machine learning where the economics permits flexibility (the mapping from bids to scores and payment scalars), but retain an explicit optimization layer where the economics demands hard guarantees (combinatorial feasibility, integrality, and auditable maximization of a declared objective). This architecture is, in our view, less a theoretical curiosity than a response

to platform reality. Real auctions are operated under latency constraints, compliance obligations, and adversarial scrutiny; they therefore benefit from mechanisms that can be *replayed*, *explained* in terms of logged artifacts, and *bounded* by interpretable approximation parameters such as solver gaps and deviation-oracle strength.

At the same time, our results should be read with the appropriate humility. The strongest guarantees we obtain "for free" are feasibility and individual rationality under truthful reporting given the payment scaling. Incentive properties, by contrast, remain approximate and distribution-dependent: regret is estimated with an attacker, and any bound inherits assumptions about solver accuracy, Lipschitzness, and the adequacy of the deviation search. This is not a defect of the approach so much as an expression of the central difficulty in the area: in rich combinatorial environments, incentives are hard to certify without reverting to restricted (and often low-revenue) classes such as affine maximizers with VCG payments. The open question is therefore not whether we can learn *something* that performs well on a training distribution, but whether we can make such performance *operationally credible* under the stresses that accompany deployment.

We conclude by outlining three next-step problems where we believe solver-backed pipelines provide a useful foundation, while also clarifying what remains technically unresolved.

(1) Scaling to large item sets and richer bidding languages. Throughout, we have described bidders as reporting a vector of values over an allowable bundle set K. This is a natural representation for analysis and for many applications where m is modest and K is engineered (e.g., a menu of feasible packages, or a set of precomputed routes). But it does not directly resolve the core scalability bottleneck of combinatorial auctions: when m is large, even representing bids over all bundles is impossible, and the interesting economic problem shifts from winner determination to *preference elicitation*.

One direction is to embed a bidding language into the mechanism and treat K as endogenous. Bidders might submit structured bids (OR-of-XOR, additive with pairwise complementarities, sparse hypergraph valuations, or ML-generated value queries), and the platform expands these into an instance-specific set of candidate bundles before solving. The mechanism then becomes a composition of (i) a representation map from messages to a bundle universe, (ii) a scorer, and (iii) a solver-backed allocation. This raises new incentive questions. If a bidder can influence which bundles enter K, then strategic behavior can occur even before the allocation step. A principled design would therefore specify K as a deterministic function of the message that is itself auditable, and it would seek message spaces where manipulation of the candidate set is either unprofitable or bounded.

A second direction is iterative interaction: rather than asking for full reports, the platform queries bidders adaptively, choosing a small set of bundles to price or evaluate. Solver-backed methods remain relevant because each iteration still requires a combinatorial optimization step under the current information set, and certificates remain meaningful. The difficulty is to reconcile iterative elicitation with incentive monitoring: regret is no longer an ex post function of a one-shot report, but a dynamic object that depends on the querying policy.

A third, more pragmatic direction is to admit that real platforms already impose strong language restrictions (budget constraints, bid caps, limited package sizes, or "business rules") and to incorporate these directly. In that case, the research goal is not to eliminate restrictions, but to understand how to choose them to optimize the revenue–robustness frontier. Solver-backed pipelines help because such restrictions can often be formalized as constraints, and thus become part of $\mathcal{X}$ (on the allocation side) or part of the admissible message space (on the reporting side), both of which are easier to document and audit than ad hoc product choices.

(2) Stronger incentive-compatibility certificates beyond empirical regret. A recurring theme is that approximate IC is only as convincing as the process used to measure it. Today, many learned mechanisms are evaluated by running a deviation oracle on samples. This is a useful engineering test, but it is not yet a certificate in the sense that a regulator, a court, or even an internal risk committee might demand. The gap between empirical regret and a deployable guarantee is therefore an important research agenda.

One promising approach is to move from *expected* regret to *instance-level* incentive diagnostics. Because the allocation step is solved (exactly or with a logged gap), we can in principle produce bounds of the following flavor: given the realized bid profile b, the logged solver gap $\eta(b)$, and a characterization of the local sensitivity of payments and scores, we can upper bound how much any bidder could have improved by deviating *within a declared deviation class*. Such bounds would not yield DSIC, but they could yield a defensible statement such as "within the allowed misreport class, no bidder can gain more than ϵ on this instance." This shifts the governance question from unverifiable global incentives to verifiable local guarantees.

A complementary approach is to restrict the mechanism class so that incentives become certifiable by construction. Proposition 5 sketches one extreme: affine maximizers with VCG payments yield DSIC, at the cost of reduced expressiveness. There may be intermediate classes that preserve some learned flexibility while admitting verification. For example, one might enforce bidder-wise monotonicity properties, or train within a maximal-in-range family where the range is explicitly enumerated and fixed, enabling truthful payments in the VCG spirit. The challenge is to find restrictions

that are not merely theoretically convenient but also computationally viable at scale.

A third approach is to improve the deviation oracle itself and, crucially, to make its limitations explicit. In practice, an "attacker" is a piece of software with a budget: it searches a parametric space of misreports, perhaps using gradient methods or heuristics. Logging the attacker configuration, its achieved utility improvements, and the residual optimization gaps can turn incentive evaluation into an auditable process. Here solver-backed deployment again helps: when winner determination emits a gap, we can propagate this gap into utility bounds (as in Proposition 3), thereby making clear how much of the uncertainty comes from optimization error rather than from strategic complexity.

(3) Online updates and distribution shift: safe learning in production. A deployed platform rarely faces a stationary environment. Bid distributions drift with seasonality, entry and exit, macro shocks, and strategic adaptation to the mechanism itself. A static θ trained offline is therefore at risk of becoming stale, while naive online updates risk introducing instability or, worse, creating opportunities for manipulation during the update process.

A key open problem is to develop *safe* update rules that preserve hard constraints (feasibility, IR-by-design) and do not materially worsen incentive properties. One operationally plausible pattern is "shadow learning": update candidate parameters using recent data while keeping the deployed mechanism fixed, then evaluate the candidate under stress tests (including stronger deviation searches) before promotion. Another is to impose explicit trust regions on parameter changes, so that the mapping $b \mapsto (s_\theta(b), \alpha_\theta(b))$ cannot shift too abruptly between versions; this connects naturally to the Lipschitz-type stability conditions that appear in regret bounds.

More ambitiously, one might seek formal guarantees under shift by coupling monitoring with re-optimization. Because solver-backed systems already log the quantities that matter—instance difficulty, objective gaps, and realized payments—they can support change-point detection and conditional retraining triggers. But the economic difficulty remains: bidders respond strategically to the mechanism, so the data-generating process depends on the policy being updated. This is an identification problem as much as an optimization problem, and it suggests that techniques from off-policy evaluation and robust reinforcement learning will need to be adapted to quasi-linear strategic environments.

Why solver-backed pipelines fit platform realities. Across these open problems, we see a consistent advantage of solver-backed design: it creates a clean interface between what is learned and what is enforced. The

optimizer enforces combinatorial constraints and produces certificates; the learned components shape objectives and payments but do not silently violate feasibility. This separation matters for governance. It allows compliance teams to reason about constraints as constraints (and to update them as policy changes), while allowing research teams to iterate on scoring architectures without rewriting the definition of $\mathcal{X}$. It also allows engineering teams to budget computational effort explicitly through the solver tolerance η, turning a latency–optimality tradeoff into a transparent mechanism parameter rather than an opaque implementation detail.

The broader implication is that accountability in modern auctions should not be framed as "learning versus theory," but as a choice of which parts of the mechanism are *specifiable, checkable, and logged*. Solver-backed learned mechanisms are attractive precisely because they make many economically meaningful claims checkable after the fact. The remaining challenge is to extend this checkability to incentives in a way that is both mathematically honest and institutionally usable. We view progress on bidding-language scalability, incentive certificates, and safe online updating as the most promising route toward that goal.