mathematics

MDPI

Review

Operational Research and Artificial Intelligence: A Functional Taxonomy of Five Integration Modes

Felipe Baesler [1,*], Eric Forcael [1], Rudy Carrasco [1], Oscar Cornejo [2] and Rodrigo Rebolledo [3]

[1] College of Engineering, Universidad San Sebastián, Concepción 4081339, Chile; eric.forcael@uss.cl (E.F.); rudy.carrasco@uss.cl (R.C.)
[2] College of Engineering and Business, Universidad de las Américas, Santiago 7500975, Chile; ocornejo@udla.cl
[3] Industrial Engineering Department, Faculty of Engineering, Universidad Católica de la Santísima Concepción, Concepción 4090541, Chile; rrebolledo@ucsc.cl
* Correspondence: felipe.baesler@uss.cl

Abstract

Artificial intelligence (AI) and operational research (OR) are increasingly combined in decision-making systems, but the literature remains fragmented across communities that classify contributions by learning technique, problem class, or solver component. This review proposes a functional taxonomy that classifies AI–OR integration according to the role that AI plays relative to the optimization process. Using an integrative review design with iterative taxonomy development, we identify five modes: predict-then-optimize, decision-focused learning, AI-enhanced optimization, learning-based optimization, and AI-augmented OR modeling and decision support. Each mode is defined formally in a common notation that specifies what is learned, which training signal is used, and which component retains decision authority at run time. These definitions are translated into an operational classification procedure, including a substitution test that separates learned guidance of OR algorithms from learned solvers, and are applied to a set of boundary cases. A cross-mode analysis compares guarantees, training signals, computational burden, and human involvement, and maps how the modes interact in composite systems. The analysis indicates that many open problems arise at the interfaces between modes, notably feasibility under uncertain constraints, error propagation in composite pipelines, semantic verification of generated models, and end-to-end evaluation. The framework provides a common vocabulary for designing and assessing hybrid AI–OR systems.

Keywords: artificial intelligence; operational research; machine learning for optimization; predict-then-optimize; decision-focused learning; neural combinatorial optimization; learning to optimize; large language models; taxonomy; hybrid decision systems

MSC: 90-02; 90C27; 90B35; 68T05; 90B50

Check for updates

Academic Editor: James Liou

Received: 11 October 2026
Revised: 28 October 2026
Accepted: 30 October 2026
Published: 14 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Artificial intelligence (AI) and operational research (OR) have long addressed complementary aspects of decision-making. AI provides methods for learning patterns, representations, and policies from data, whereas OR provides mathematical models and algorithms for selecting actions under objectives and constraints. The availability of data, advances in machine learning, and the growing complexity of operational systems have brought these traditions into close interaction, producing a rapidly expanding body of research at the intersection of learning and optimization.

This convergence is visible across several previously distinct streams. Predictive models are routinely used to estimate uncertain parameters before optimization; machine-learning components are embedded within mixed-integer and heuristic solvers; decision-focused learning trains predictors using downstream decision quality [1]; neural combinatorial optimization learns policies that construct or improve solutions directly [2]; and large language models (LLMs) are used to formulate mathematical programs, generate solver code, verify formulations, and mediate interaction with decision-makers [3,4].

Several surveys have organized parts of this landscape. Fan et al. [5] review AI for optimization across the stages of the optimization process; Bengio et al. [6] review machine learning for combinatorial optimization from a solver-centered perspective; Mandi et al. [1] and Sadana et al. [7] cover decision-focused and contextual optimization on the prediction side; Fajemisin et al. [8] survey optimization with learned constraints; and Angioni et al. [2] provide a tutorial on neural combinatorial optimization. Each of these works covers one segment of the pipeline and adopts the vocabulary of its community. As a result, closely related integration mechanisms are described with different terminology, while methods that use the same AI technology may play fundamentally different roles in the decision process.

This fragmentation creates two problems for a review that aims to span the field. First, technology-centered classifications become unstable as the same architectures migrate across stages of the OR pipeline: a graph neural network, a reinforcement-learning policy, or an LLM can support internal solver decisions, generate complete solutions, formulate models, or explain results. Second, the boundaries between neighboring paradigms are often unclear. A learned component inside a heuristic may either assist an OR-controlled search or become the principal solving mechanism; prediction followed by optimization may correspond either to conventional predict-then-optimize or to decision-focused learning, depending on how the predictor is trained.

To address these issues, this review proposes a functional taxonomy of AI–OR integration. Rather than asking which AI technology is used, the framework asks what function AI performs relative to OR, what element is learned or transformed, how it is trained, and which component retains control over the generation of the decision. On this basis, five integration modes are identified: (1) predict-then-optimize, (2) decision-focused learning, (3) AI-enhanced optimization, (4) learning-based optimization, and (5) AI-augmented OR modeling and decision support. The modes classify integration mechanisms rather than entire applications: a single system may predict demand, use learned guidance inside an OR solver, and explain the optimized decision through an LLM interface.

The review makes four contributions. First, it develops a five-mode taxonomy that connects the prediction-side, solver-side, and modeling-side literatures within a single framework and defines each mode formally in a common notation. Second, it turns the taxonomy into an operational classification procedure, including a decision tree, a substitution test for the boundary between learned guidance and learned solvers, and an analysis of boundary cases. Third, it compares the modes across decision authority, dependence on OR structure, training signals, computational burden, guarantees, and human involvement, and maps how the modes interact in composite systems. Fourth, it derives a research agenda from the formal framework, centered on feasibility under uncertain constraints, error propagation across components, semantic verification, human interaction, and end-to-end evaluation.

The scope is deliberately broad but functional rather than exhaustive. The objective is not to catalog every paper on machine learning for optimization, but to identify the main mechanisms through which AI and OR are integrated and to examine representative and influential works that clarify their evolution and boundaries. This review addresses the

operational research community: it examines how AI mechanisms attach to the models, algorithms, and decision processes that this community has developed.

The remainder of the paper is organized as follows. Section 2 positions the review with respect to existing surveys and taxonomies. Section 3 describes the review methodology. Section 4 introduces the formal framework, the classification procedure, and the boundary cases. Section 5 reviews the literature within each mode. Section 6 compares the modes and analyzes their interactions. Section 7 identifies research gaps, Section 8 discusses implications and limitations, and Section 9 concludes.

2. Related Reviews and Existing Taxonomies

Existing reviews of the intersection between learning and optimization can be grouped into four families according to the segment of the decision pipeline they cover. Table 1 summarizes representative works, their organizing criterion, and their relation to the present review.

Table 1. Representative reviews and taxonomies at the intersection of learning and optimization.

Review	Scope	Organizing Criterion	Relation to This Review
Fan et al. [5]	AI across the optimization process	Stage of the process: parameter generation, model formulation, solution methods, solution interpretation	Closest in breadth; stage-based rather than functional
Bengio et al. [6]	Machine learning for combinatorial optimization	End-to-end learning; learning to configure algorithms; ML alongside optimization algorithms	Closest antecedent of the M3/M4 boundary; solver side only
Talbi [9]; Karimi-Mamaghan et al. [10]	Machine learning in metaheuristics	Level at which learning enters the metaheuristic	Refines part of M3 (learning-guided search)
Mazyavkina et al. [11]; Cappart et al. [12]	Reinforcement learning and graph neural networks for combinatorial optimization	Learning technology	Technology-centered; spans M3 and M4
Angioni et al. [2]	Neural combinatorial optimization	Solution construction and improvement	Core of M4
Kotary et al. [13]	End-to-end constrained optimization learning	Learning solutions versus optimization inside learning	Spans M2 and M4
Mandi et al. [1]	Decision-focused learning	Mechanism used to obtain training gradients	Core of M2
Sadana et al. [7]	Contextual optimization under uncertainty	Decision rules; sequential learning and optimization; integrated learning and optimization	Corresponds to M1, M2, and part of M4
Fajemisin et al. [8]	Optimization with constraint learning	Framework for embedding learned constraints	Part of M1 (learned model components)
Mitzenmacher and Vassilvitskii [14]	Algorithms with predictions	Guarantees under prediction error	Cross-cutting paradigm within M3
Xiao et al. [15]	LLM-based optimization modeling	Technical stack: data synthesis and fine-tuning, inference frameworks, benchmarks, evaluation	Part of M5 (modeling and verification)

The broadest recent survey, by Fan et al. [5], organizes AI for optimization by the stage of the optimization process at which AI intervenes: parameter generation, model formulation, solution methods, and solution interpretation. The first family is solver-centered. Bengio et al. [6] distinguish methods that learn to solve problems end-to-end, methods that learn to configure algorithms, and methods in which learning is invoked repeatedly alongside an optimization algorithm. Surveys of learning in metaheuristics [9,10] classify contributions by the level at which learning enters the search, and technology-centered surveys organize the field by reinforcement learning or graph neural networks [11,12]. The second family is prediction-centered: Mandi et al. [1] organize decision-focused learning by how gradients through the optimization problem are obtained, and Sadana et al. [7] distinguish decision rules, sequential learning and optimization, and integrated learning and optimization in contextual problems. The third family concerns optimization models that contain learned components, such as constraint learning [8] and algorithms with predictions [14]. The fourth, and most recent, concerns LLM-assisted formulation, verification, and interaction, which is largely absent from the earlier surveys and is reviewed by Xiao et al. [15].

The present review differs from these works in four respects. First, it places the prediction, solving, and modeling layers within one framework, so that mechanisms from different communities can be compared on common dimensions. Second, its organizing criterion is functional: modes are defined by what is learned, how it is trained, and which component governs the decision at run time, not by the learning technology. The stage-based organization of Fan et al. [5] is close to the layer dimension D1 of Section 4.2, but a stage alone places predict-then-optimize and decision-focused learning under the same heading (parameter generation) and learned guidance of solvers and learned solvers under another (solution methods); the coupling and authority dimensions used here separate these cases. Third, it gives explicit formal definitions of each mode and an operational classification procedure. The M3/M4 distinction is related in spirit to the distinction by Bengio et al. [6] between learning alongside algorithms and end-to-end learning; the contribution here is to make it testable through a substitution criterion and to extend it to offline algorithm design and LLM-generated heuristics. Similarly, the M1/M2 distinction corresponds to the sequential-versus-integrated distinction of Sadana et al. [7]. The treatment of decision rules differs, however: because a learned decision rule outputs the decision directly, the authority criterion used here places it with learned solvers in M4 rather than in a separate category. Fourth, the review adds a modeling and interaction layer (M5) and analyzes how the five modes interact within a single system.

3. Review Methodology

3.1. Review Design

This study adopts an integrative review design combined with taxonomy development. The literature relevant to AI–OR integration spans predictive modeling, mathematical optimization, solver learning, decision-focused learning, neural combinatorial optimization, large language models, and interactive decision support, each with partially distinct terminology and publication venues. The objective is therefore to synthesize integration mechanisms across communities, not to estimate a pooled effect or to enumerate every publication. The review is not presented as a systematic review, scoping review, or meta-analysis, and it does not claim conformity with PRISMA or its scoping extension. Instead, it reports its search and coding procedures transparently so that the classification can be scrutinized and extended.

The unit of analysis is the AI–OR integration mechanism described in each study. Papers were included when they provided a substantive methodological contribution,

benchmark, survey, or representative application that clarified how AI interacts with an OR model, algorithm, solution process, or decision-support interface.

3.2. Taxonomy Development Procedure

The taxonomy was developed following the method of Nickerson et al. [16], which combines empirical-to-conceptual iterations (examining objects and grouping their characteristics) with conceptual-to-empirical iterations (deriving dimensions from theory and testing them against objects). The meta-characteristic, that is, the purpose that determines which characteristics are relevant, is the function performed by AI relative to the OR decision process.

Initial dimensions were derived conceptually from the structure of an optimization-based decision pipeline: where learning enters, what it produces, how it is trained, and which component retains control over the decision. These dimensions were then tested empirically against the coded studies. Iterations continued until the following ending conditions held: every coded mechanism could be assigned to at least one mode; every pair of modes differed in at least one dimension; no dimension or mode was added, merged, or split in the last iteration; and the difficult cases listed in Section 4.5 could be resolved by the classification procedure without ad hoc rules. Established labels were retained where they correspond to coherent research traditions (predict-then-optimize, decision-focused learning), and broader functional labels were used when previously separate streams share the same mechanism. The resulting taxonomy is therefore a theory-informed functional synthesis tested against the literature, not a statistical clustering of papers.

3.3. Literature Identification

Initial searches were organized around established and emerging labels: predict-then-optimize, decision-focused learning, contextual optimization, constraint learning, learning to optimize, learning to branch, learning-augmented algorithms, algorithm selection and configuration, hyper-heuristics, neural combinatorial optimization, learning-guided heuristics, optimization modeling with large language models, and explainable or interactive operational research. These labels guided an initial, concept-driven identification of the literature, which was then complemented by a structured search. The structured search was conducted in Scopus, selected for its broad coverage of both operational research journals and computer science conference proceedings, using nine queries that correspond to the streams reviewed in Section 5. The queries, the dates on which they were last run, and the number of records they returned are reported in Appendix A.

Because terminology is fragmented across communities, keyword searches were complemented by backward and forward snowballing following Wohlin [17]. Surveys and tutorials were used as orientation sources to identify foundational studies, competing terminology, and methodological branches; primary papers were then examined to verify the mechanism attributed to them. When a candidate paper exposed a distinct mechanism, a missing lineage, or a boundary case, additional targeted searches were performed. Searching by labels carries a risk of circularity, since the labels partly anticipate the modes; the conceptual derivation of the dimensions (Section 4.2) and the boundary-case analysis (Section 4.5) were used to counter this risk.

Eligible studies contained both a recognizable learning component and a substantive OR component (mathematical programming, combinatorial optimization, search, heuristics, dynamic programming, constraint-based reasoning, or formal decision modeling), and the interaction between them was central to the contribution. Throughout the paper, "AI" refers to learned components, including machine-learning models and LLMs. Stand-alone metaheuristics without a learned component, fuzzy systems, and traditional expert systems

were excluded, as were symbolic AI methods such as constraint programming or SAT solving when they act as the OR component rather than as a learned one. The nine queries returned 11,004 records in total; 9488 were screened by title, abstract, and venue, the remainder corresponding to the low-relevance tails of the largest query, for which the most cited and the most recent records were screened instead (Appendix A). Screening led to 30 studies being added to the review. The structured search also served as a coverage check: it recovered the central references of each stream that had been identified earlier, which supports the adequacy of the initial identification, while confirming that work published in machine learning proceedings series is unevenly indexed and is reached mainly through snowballing. The counts reported in Appendix A are query-level totals: a record returned by more than one query is counted once per query, and no de-duplicated union of the nine result sets was computed, because screening was carried out query by query in order to measure the coverage of each stream separately. The two right-hand columns of Table A1 separate coverage from yield: across the nine queries, 42 recoveries correspond to studies already identified by the concept-driven search and snowballing, and 30 studies were added to the review. The final corpus contains 109 unique references, de-duplicated in the reference manager, of which 99 are coded mechanisms and 10 serve another function in the article.

3.4. Coding Scheme

For each included study, the following attributes were recorded: the layer at which AI intervenes (D1), the object learned or transformed (D2), the training signal (D3), the component with decision authority at run time (D4), the role of the human (D5), the methodological subfamily, the boundary with neighboring modes, and the role of the study in the development of its stream (precursor, foundational contribution, consolidation, survey, or recent frontier). Dimensions D1–D5 are defined in Section 4.2. The coding was performed by the first author, who assigned a mode to every included study; the co-authors reviewed the assignments, and uncertain cases and disagreements were resolved by discussion among the authors. No independent double-coding was carried out, and no inter-rater agreement statistic is reported; independent coding by a second rater remains a natural validation step for future work. The taxonomy stabilized after three iterations of the procedure as described in Section 3.2. The resulting distribution comprises 8 studies in M1, 21 in M2, 30 in M3, 17 in M4, and 20 in M5, with 3 further studies reported as borderline between M3 and M4 (Section 4.5), giving 99 coded mechanisms; the remaining ten references are seven prior reviews and taxonomies, two methodological references, and one supporting reference from outside the operational research literature. Dimensions D1 and D4 follow from the assigned mode by construction, since Table 2 defines the modes as combinations of those two dimensions, whereas D2, D3, and D5 were recorded at the level of the methodological subfamily. The complete coding matrix, with the dimension values and the study-specific exceptions, is provided as Supplementary Table S1.

3.5. Evidence Use and Bibliographic Verification

Primary sources were preferred when describing the technical mechanism of a method. Surveys and tutorials were used for historical context, terminology, and coverage, but claims about individual algorithms were checked against the original publication whenever possible. This distinction matters because high-level summaries can obscure technical differences that affect classification, for example whether a method differentiates through an optimizer, invokes a solver during training, or merely uses optimization outputs as supervision. The reference list contains only works cited in the manuscript, not the complete set of candidate studies examined.

3.6. Limitations of the Design

Purposive searching and snowballing can under-represent peripheral communities whose vocabulary differs from the search terms, and classification involves analytical judgment in hybrid cases. The structured search relied on a single database, so studies indexed only elsewhere may have been missed; this is mitigated, but not removed, by the snowballing and by the direct examination of the proceedings of the main venues in the field. These limitations are mitigated by explicit formal definitions, a documented classification procedure, repeated examination of counterexamples, reliance on primary sources, and the public coding matrix. Further limitations of the framework itself are discussed in the "Limitations" portion of Section 8. In particular, the search terms favor the communities that use them, and adjacent communities such as robust optimization, stochastic programming, control theory, and human–computer interaction may describe functionally related mechanisms with a different vocabulary; claims about the literature should be read accordingly.

4. A Formal Functional Framework

4.1. Notation and Running Example

We consider a decision problem parameterized by a vector of possibly unknown quantities θ:

$$P(\theta): \quad \min_{z \in Z(\theta)} f(z;\theta), \qquad z^*(\theta) \in \arg\min_{z \in Z(\theta)} f(z;\theta), \tag{1}$$

where z is the decision, f the objective, and $Z(\theta)$ the feasible set. Contextual information x is available before the decision, historical data take the form $\mathcal{S} = \{(x_i, \theta_i)\}_{i=1}^{N}$, and $\mathcal{D}_\Xi$ denotes the distribution from which instances are drawn. An instance is denoted $\xi = (f, Z, \theta)$, and an algorithm A maps instances to decisions, $z = A(\xi)$. Before any of this exists, the problem is available only as a description d (text, rules, stakeholder goals); a modeling map $d \mapsto (f, Z)$ produces the formal model. Learned components carry parameters w (predictors), ϕ (solver-side components), or φ (modeling and interaction components). Two conventions are used throughout: when the feasible set does not depend on θ, we write Z instead of $Z(\theta)$, and when the instance is the natural argument, we write $f(z;\xi)$ for $f(z;\theta)$ with $\xi = (f, Z, \theta)$. The notation is summarized in Appendix B.

Example 1 (**weekly surgical scheduling**). *To illustrate the framework, we use a stylized example; it is not a description of an implemented system or of a study reviewed here. The example is motivated by the authors' ongoing work on AI-supported elective surgical scheduling in a public hospital, which explores the integration of several of the modes discussed in this review; no data or results from that project are reported here. Elective cases j must be assigned to operating theaters r and days t. The decision $z_{jrt} \in \{0,1\}$ indicates an assignment; θ_j is the duration of case j; x contains patient, procedure, and surgeon features; and the feasible set includes the capacity constraints*

$$\sum\nolimits_j \theta_j z_{jrt} \le C_{rt} \qquad \forall\, r, t, \tag{2}$$

where C_{rt} is the session length. The durations are uncertain and enter the constraints, not only the objective.

4.2. Classification Dimensions

Five dimensions structure the framework.

- **D1. Layer of intervention.** AI may act on the *parameter layer* (the inputs or components of P), on the *solution-process layer* (how z is obtained from P), or on the *model and interface layer* (how d becomes (f, Z) and how results reach the decision-maker).

- **D2. Object learned or transformed.** Parameters, distributions, or model components; decision rules inside an algorithm; algorithm configurations or code; solution generators; formulations, preferences, or explanations.
- **D3. Training signal.** A statistical loss independent of P; a decision loss computed through P; a performance measure of an algorithm; the objective value of generated solutions; or semantic and solver-based feedback on formulations.
- **D4. Decision authority at run time.** Which component produces z: an OR solver, an OR algorithm with learned components, a learned model, or a human–model loop.
- **D5. Role of the human.** Outside the computational loop, or a participant who formulates, validates, queries, or steers the model.

Crossing D1 with the degree of coupling between learning and optimization yields the grid in Table 2. In each layer, one mode keeps learning and optimization loosely coupled and one couples them tightly, either through training (M2) or by transferring authority to the learned component (M4). In the model and interface layer, the same distinction separates assistive from autonomous systems, which we treat as two variants of M5 rather than as separate modes, because both act on the representation of the problem, not on its numerical solution.

Table 2. The five modes as combinations of layer of intervention and degree of coupling.

Layer of Intervention	Loose Coupling: OR Retains Control	Tight Coupling
Parameter layer	M1 Predict-then-optimize: predictor trained independently of P	M2 Decision-focused learning: P shapes predictor training
Solution-process layer	M3 AI-enhanced optimization: learned rules inside an OR master algorithm	M4 Learning-based optimization: learned model generates the solution
Model and interface layer	M5 (assistive): AI supports formulation, verification, and interaction under human validation	M5 (autonomous): agents formulate, solve, and debug with limited human validation

Figure 1 places the modes along the decision pipeline.

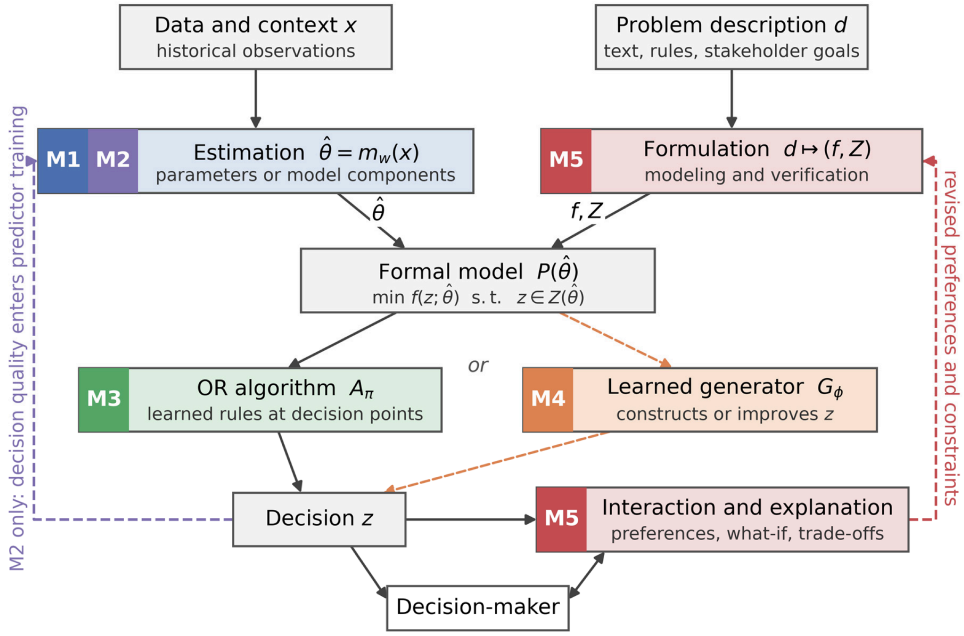


Figure 1. The AI–OR decision pipeline and the points at which the five integration modes intervene. Dashed paths indicate the decision-aware training signal of M2, the alternative solution path of M4, and the feedback of revised preferences in M5. Each mode is identified by its colour and by its label (M1 to M5) at the point of the pipeline where it intervenes.

4.3. Formal Definitions of the Modes

Mode 1: Predict-Then-Optimize (M1). A predictor m_w is fitted with a statistical loss ℓ, and its output is passed to the optimization problem:

$$\hat{w} \in \operatorname*{argmin}_w \frac{1}{N}\sum_{i=1}^{N} \ell(m_w(x_i), \theta_i), \qquad \hat{\theta}(x) = m_{\hat{w}}(x), \qquad \hat{z}(x) = z^*\big(\hat{\theta}(x)\big). \tag{3}$$

The defining property is that P does not appear in the training objective. The output of m_w need not be a point estimate. It may be a set of nonnegative weights $\omega_i(x)$ over historical observations, summing to one, that defines a conditional sample-average problem over a feasible set Z that does not depend on θ [18],

$$\hat{z}(x) \in \operatorname*{argmin}_{z\in Z} \sum_{i=1}^{N} \omega_i(x) f(z;\theta_i), \qquad \omega_i(x) \geq 0, \quad \sum_{i=1}^{N} \omega_i(x) = 1, \tag{4}$$

a learned uncertainty set for robust optimization, or a learned function embedded in the model as a constraint or objective term (constraint learning). All remain in M1 as long as the learned component is trained without reference to the downstream decision. In Example 1, M1 corresponds to fitting a duration regressor and solving the scheduling model with the predicted durations.

Mode 2: Decision-Focused Learning (M2). The predictor is trained with a loss evaluated through the optimization problem:

$$\hat{w} \in \operatorname*{argmin}_w \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}\big(z^*(m_w(x_i)); \theta_i\big), \tag{5}$$

typically the regret of the induced decision under the true parameters,

$$\mathrm{Regret}\big(\hat{\theta}, \theta\big) = f\big(z^*\big(\hat{\theta}\big); \theta\big) - f\big(z^*(\theta); \theta\big) \geq 0. \tag{6}$$

Inference is identical to M1; only the training objective differs (Figure 2). Gradient-based training requires the chain

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial z^*}\,\frac{\partial z^*}{\partial \hat{\theta}}\,\frac{\partial \hat{\theta}}{\partial w}, \tag{7}$$

whose middle factor is the source of the technical difficulty: in the setting of interest here, where the predicted parameters enter only the objective coefficients, the feasible set is fixed and independent of them, and a selection rule resolves the argmin when it is not a singleton, in that setting, $z^*\big(\hat{\theta}\big)$ is piecewise constant, so its Jacobian is zero almost everywhere and undefined elsewhere. The statement is specific to that setting: it describes neither problems in which parameters enter the constraints nor every differentiable optimization layer, since a strictly convex parametric program has a well-defined solution Jacobian. The families reviewed in Section 5.2 differ in how they handle this factor. Smoothing approaches replace z^* by a differentiable surrogate, for instance by adding a regularization term [19] or by averaging over random perturbations [20]; Equation (7) should therefore be read as a formal expression that motivates the families below, and not as a well-defined gradient for the problems of interest. In Equation (8), the expectation is taken over the zero-mean random vector, not over the perturbation scale; when the perturbation admits a positive, differentiable density on the relevant space, as in the Gaussian and Gumbel instantiations used in practice, and the feasible set is compact, the perturbed map is Lipschitz and differentiable almost everywhere, which is what makes the surrogate trainable.

$$z^*_\varepsilon\big(\hat{\theta}\big) = \mathbb{E}_\zeta\big[\, z^*\big(\hat{\theta} + \varepsilon\zeta\big)\,\big]. \tag{8}$$

Surrogate-loss approaches avoid the factor altogether. For linear objectives $f(z;c) = c^\top z$, the SPO+ loss of Elmachtoub and Grigas [21] is

$$\ell_{\mathrm{SPO+}}(\hat{c}, c) = \max_{z\in Z}\big\{(c - 2\hat{c})^\top z\big\} + 2\hat{c}^\top z^*(c) - c^\top z^*(c), \tag{9}$$

a convex upper bound on regret that admits subgradients computed by solving an optimization problem with modified costs. Gradient-free approaches search directly over predictor structures using decision-based evaluations. The formal development assumes a nonempty, compact, and convex feasible set, which does not exclude combinatorial or mixed-integer problems: for a linear objective, the feasible set can be described implicitly as the convex hull of the discrete alternatives, over which the optimum is attained at the same points. The practical cost is that evaluating the maximum, and hence computing a subgradient, requires a solver call on the original problem.

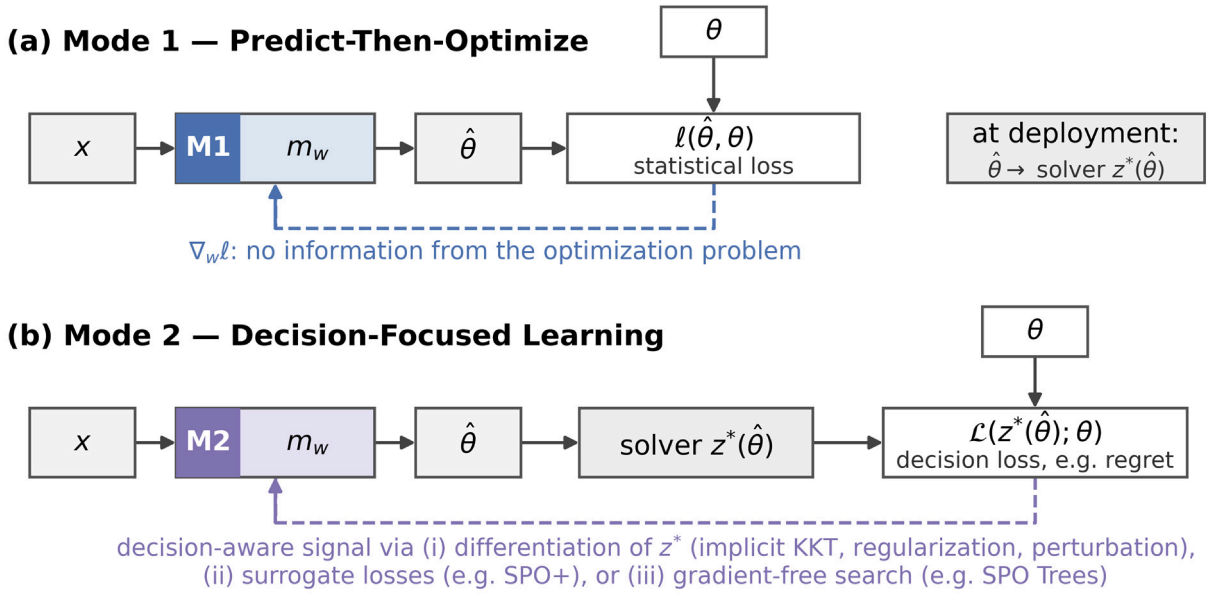


Figure 2. Information flow during training in M1 and M2. Both modes share the deployment pipeline $\hat{\theta} \rightarrow z^*(\hat{\theta})$; they differ only in whether the optimization problem contributes to the training signal of the predictor.

Equations (5) and (6) presume that Z does not depend on θ. When uncertain parameters enter the constraints, as in Equation (2), the induced decision may be infeasible for the true parameters, and regret alone no longer measures decision quality. Writing $Z(\theta) = \{z : g(z; \theta) \leq 0\}$, a natural extension penalizes violations,

$$\mathcal{L}(\hat{z}; \theta) = \left[f(\hat{z}; \theta) - f\left(z^*(\theta); \theta\right) \right]_+ + \lambda \, \| \left[g(\hat{z}; \theta) \right]_+ \|_1, \tag{10}$$

where $[\cdot]_+$ denotes the positive part and $\lambda > 0$ prices infeasibility. Because an infeasible $\hat{z}$ can attain a better objective value than $z^*(\theta)$, the raw difference between the two objective values may be negative; the positive part on the first bracket prevents the loss from rewarding infeasibility on that account. The weight λ still governs the trade-off between suboptimality and violation. For the ordinary objective-plus-penalty function, that is, without the positive part on the first bracket, exact-penalty theory relates for convex problems the threshold above which the penalized and the constrained problems share a minimizer to the optimal Lagrange multipliers of the constraints. The clipping in Equation (10) changes the function, and that threshold characterization does not carry over to it: the positive part is a safeguard against rewarding infeasibility, not a device that inherits an exact-penalty guarantee. Equation (10) is illustrative. Section 5.2.3 reviews concrete realizations of this idea, and Section 7.2 discusses what remains open. In Example 1, M2 corresponds to training the duration predictor on the scheduling cost, including overtime caused by under-predicted durations.

Mode 3: AI-Enhanced Optimization (M3). Consider an algorithm with decision points: at step t, it is in state s_t and takes an action $a_t = \pi(s_t)$ from an admissible set, such as the branching variable in branch-and-bound or the repair operator in large neighborhood search. Let Π be the class of admissible rules and $A_\pi(\xi)$ the output of the algorithm when rule π is used.

Definition 1 (OR master algorithm). *A is an OR master algorithm for a class of instances Ξ* if *(i) for every $\pi \in \Pi$ and $\xi \in \Xi$, $A_\pi(\xi)$ terminates with an output satisfying the validity properties of A (feasibility of the returned solution and, if A is exact, an optimality certificate), and (ii) for some non-learned default rule $\pi_0 \in \Pi$, A_{π_0} is an established OR method. A method counts as established when it is an autonomous, recognized OR procedure that governs the search independently of the learned component and carries validity properties of its own; recognized families include branch-and-bound, cutting-plane methods, large neighborhood search, dynamic programming, constraint programming, and the Lin–Kernighan–Helsgaun heuristic, together with their classical variants. This list is illustrative rather than exhaustive: a newly proposed procedure qualifies on the same terms, provided that the validity property invoked in condition (i) is stated explicitly. Conditions (i) and (ii) answer different questions. Condition (i) is a validity test and can be checked without the learned component: with the default rule, does the algorithm terminate with the stated property? Condition (ii) is a test of precedent: is the algorithm with that default rule a recognized procedure that governs the search on its own? The residual ambiguity of the criterion lies in (ii), which is a judgment about the state of the field rather than a mathematical property, and that is why the cases in Section 4.5 remain borderline. The class of admissible rules is likewise not arbitrary: a rule is admissible when, at each decision point, its action set is restricted to actions whose use preserves the validity property invoked in condition (i). The restriction is specific to the host algorithm, and stating it is what makes condition (i) verifiable rather than assumed. Selecting among cuts that are already valid for the formulation is admissible in a cutting-plane method, whereas generating unrestricted cuts is not, since an invalid cut can remove optimal solutions. Likewise, a learned rule that restricts the candidate solutions or the search states the algorithm may visit can preserve the feasibility of whatever is returned while forfeiting completeness, and is therefore admissible only with respect to the weaker property.*

M3 comprises mechanisms that learn a rule, configuration, or design for an OR master algorithm:

$$\hat{\phi} \in \operatorname*{argmin}_{\phi} \mathbb{E}_{\xi \sim \mathcal{D}_\Xi}\Big[\, C\big(A_{\pi_\phi}, \xi\big)\,\Big], \qquad \pi_\phi \in \Pi, \tag{11}$$

where C is a performance measure such as run time, number of nodes, primal integral, or optimality gap. Configuration and selection act before the run, by choosing a configuration $\kappa \in \mathcal{K}$ for each instance through a learned performance model $\hat{C}_\phi$:

$$\hat{\kappa}(\xi) \in \operatorname*{argmin}_{\kappa \in \mathcal{K}} \hat{C}_\phi(\kappa, \xi). \tag{12}$$

In LLM-based algorithm design, κ is program code and the search over $\mathcal{K}$ is carried out by a generative model. By Definition 1 (i), learning in M3 affects performance but not the validity properties of the master algorithm, and which properties these are depends on that algorithm: a learned branching or cutting rule inside branch-and-bound preserves feasibility, termination, completeness, and the optimality certificate, whereas a learned operator inside a heuristic such as large neighborhood search preserves the feasibility of the returned solution and the termination criterion of the host procedure, which carries no optimality guarantee to begin with. These properties are separable and should be claimed individually: feasibility of the returned solution, termination, completeness of the search, and certification of optimality. A learned restriction of the search space, for instance, may

preserve feasibility and termination while forfeiting completeness. One qualification applies to every exact method executed under a time limit: what is returned is a feasible incumbent together with an optimality gap, so the certificate is conditional on the algorithm running to completion, and reported results should state which of the two situations they describe. The inherited guarantees are those of the master algorithm and should be stated as such rather than assumed uniform across M3. Learning-augmented algorithms make this explicit through guarantees of the schematic form

$$\mathrm{ALG}\big(\xi;\hat{\theta}\big) \;\le\; \min\Big\{\alpha\,\mathrm{OPT}(\xi) + \beta\,\eta\big(\hat{\theta},\theta\big),\; \gamma\,\mathrm{OPT}(\xi)\Big\}, \tag{13}$$

where $\hat{\theta}$ is the prediction supplied to the algorithm, $\eta\big(\hat{\theta},\theta\big) \ge 0$ measures its error, and $\alpha \le \gamma$ and β are problem-dependent constants: performance is near-optimal when predictions are accurate (consistency, governed by α) and never worse than a fixed factor γ when they are not (robustness). Published bounds take problem-specific functional forms, for instance depending on the ratio of prediction error to the optimal cost; Equation (13) conveys only their common structure. In Example 1, M3 corresponds to a learned branching rule or primal heuristic inside the MIP solver; the optimality certificate of branch-and-bound is unaffected.

Mode 4: Learning-Based Optimization (M4). A learned model generates the solution:

$$z = G_\phi(\xi), \qquad \hat{\phi} \in \operatorname*{argmin}_{\phi} \mathbb{E}_{\xi\sim\mathcal{D}_\Xi}\Big[f\big(G_\phi(\xi);\xi\big)\Big]. \tag{14}$$

In constructive methods, G_ϕ is a sequential policy, and $p_\phi(z \mid \xi) = \prod_t \pi_\phi(a_t \mid s_t)$, often trained with a policy-gradient estimator with baseline $b(\xi)$. The operational difference with M2 is that, in M2, the learned predictor supplies inputs to a solver and the solver produces the decision, whereas in M4, the learned generator produces the decision itself; both train on a decision-dependent signal, but only M4 assigns decision authority to the learned component at run time.

$$\nabla_\phi J(\phi) = \mathbb{E}_{z\sim p_\phi(\cdot|\xi)}\Big[\big(f(z;\xi) - b(\xi)\big)\,\nabla_\phi \log p_\phi(z \mid \xi)\Big]. \tag{15}$$

Improvement methods iterate a learned operator, $z^{(m+1)} = I_\phi\big(z^{(m)};\xi\big)$; amortized methods map instances to solutions in a single pass; and decision rules map context directly to decisions, $z = q_\phi(x)$, trained on empirical decision cost. The defining property is that no OR master algorithm in the sense of Definition 1 governs solution generation; OR structure may still supply feasibility masks, rewards, repair operators, teachers, and benchmarks (see Section 4.4). In Example 1, M4 corresponds to a policy that repairs the daily schedule after cancellations in milliseconds.

Mode 5: AI-Augmented OR Modeling and Decision Support (M5). AI acts on the modeling map and on the interface with the decision-maker. In *modeling*, a learned map produces a formulation, or executable code, from a description, $\big(\hat{f},\hat{Z}\big) = M_\varphi(d)$. In *verification*, the generated model is compared with an intended model (f, Z). A reduction-type criterion, inspired by Karp reductions, requires an efficiently computable map ψ from the solutions of the generated model to those of the intended model that preserves feasibility and optimality:

$$\psi(\hat{z}) \in Z \;\text{ for every } \hat{z} \in \hat{Z}, \qquad \psi\big(\hat{z}^*\big) \in \operatorname*{argmin}_{z\in Z} f(z). \tag{16}$$

The first condition of Equation (16) states that every feasible solution of the generated model is mapped to a feasible solution of the intended model, so the map acts on solutions rather than on feasible sets, and it is required to be efficiently computable. The second condition is required for every optimal solution of the generated model, not only for the one a particular solver happens to return, which matters when alternative optima exist; Equation (16) therefore establishes a solution-transfer property and does not assert equivalence of

the feasible sets or preservation of objective values. It is also one-directional; equivalence notions differ in whether such maps are required in both directions, in the class of admissible maps, and in whether they are checked on particular instances or for all instances of a formulation [22,23]. In *interaction*, the model depends on preference or constraint parameters ρ that are updated from the decision-maker's responses r_k, interpreted by a learned component U_φ:

$$z_k \in \arg\min_{z \in Z(\theta,\rho_k)} f(z;\theta,\rho_k), \qquad \rho_{k+1} = U_\varphi(\rho_k, r_k). \tag{17}$$

M5 is *assistive* when a human validates the output of M_φ or U_φ within the loop, and *autonomous* when an agent formulates, solves, and debugs with limited human validation. In Example 1, M5 corresponds to translating planners' rules on surgeon availability into verified constraints and to explaining why a case was moved to another day.

Table 3 summarizes the definitions.

Table 3. Formal summary of the five integration modes.

Mode	Object Learned	Training Signal	Decision Authority at Run Time	Validity Inherited from
M1 Predict-Then-Optimize	m_w: parameters, distributions, uncertainty sets, or model components	Statistical loss, Equation (3)	OR solver	Solver, for the predicted instance
M2 Decision-Focused Learning	m_w	Decision loss through P, Equations (5)–(10)	OR solver	Solver, for the predicted instance
M3 AI-Enhanced Optimization	Rules π_ϕ, configurations κ, or code for an OR master algorithm	Algorithm performance, Equations (11) and (12)	OR master algorithm	Master algorithm, Definition 1 (i)
M4 Learning-Based Optimization	Generator G_ϕ, improvement operator, or decision rule	Solution quality: supervised imitation of solver-generated solutions, or reward-based training, Equations (14) and (15)	Learned model	Masks and repair only, if present
M5 AI-Augmented Modeling and Decision Support	Modeling map M_φ, preference update U_φ, explanations	Semantic, solver-based, or interaction feedback	Usually OR solver; human–model loop	Verification criterion, e.g., Equation (16)

4.4. Operational Classification Procedure

The definitions are turned into the decision tree in Figure 3. The procedure is applied to one mechanism at a time; composite systems are decomposed, and each component is classified separately. The order of the questions matters. Q1 is asked first because modeling and interaction mechanisms may involve predictors and solvers as subcomponents, and the question is what the AI component under analysis acts on. Q2 precedes Q3 because a component that produces model inputs is classified by its training signal (Q2a), regardless of which solver is used downstream. Q3a implements Definition 1 as a *substitution test*: the learned component is replaced by a default non-learned rule, and the mechanism belongs to M3 only if the remaining procedure is still valid for any admissible rule and is itself an established OR method.

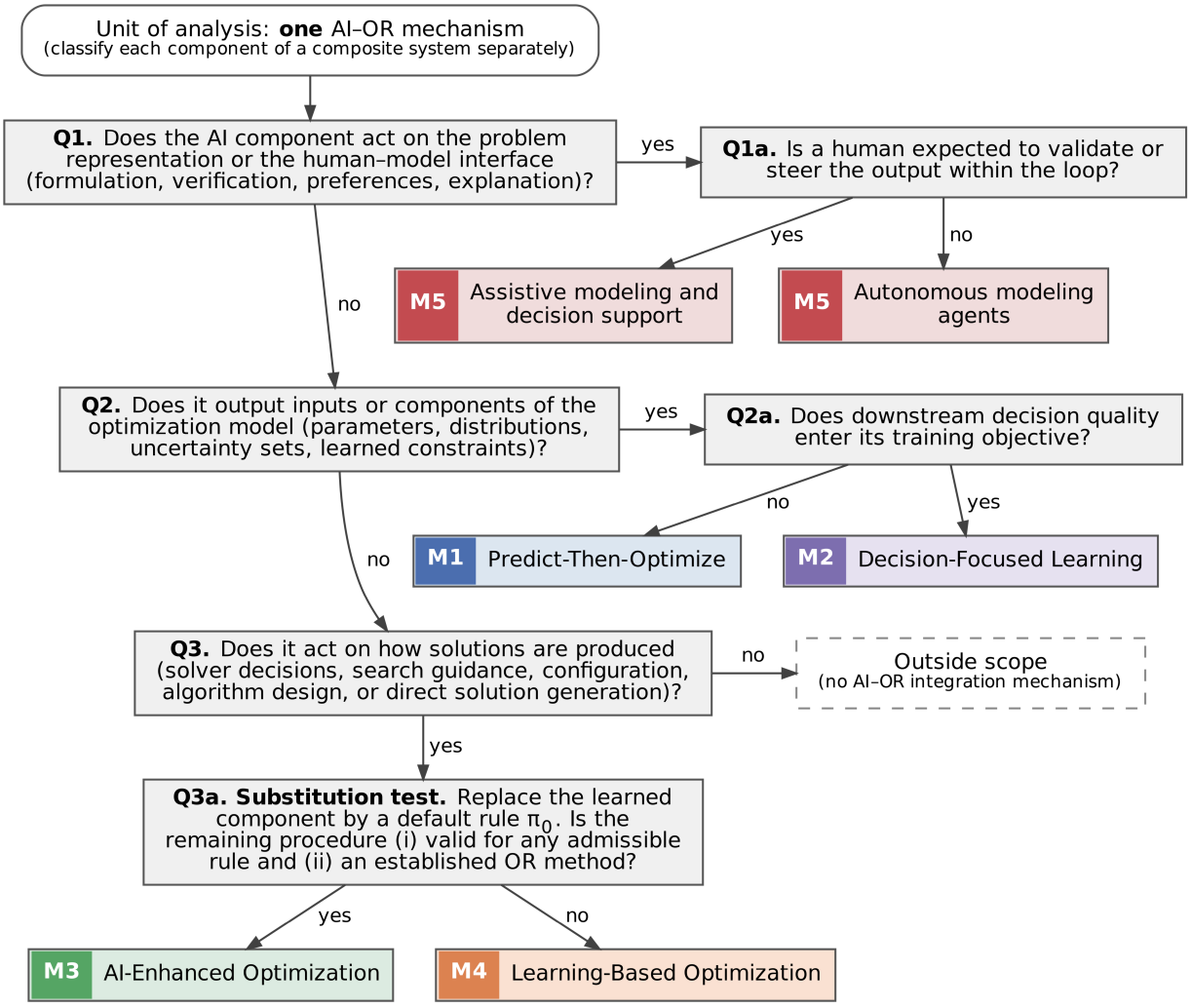


Figure 3. Decision tree for classifying an AI–OR integration mechanism. Q3a implements the substitution test of Definition 1.

Both conditions of the test are needed. Condition (i) alone would place in M3 any learned policy wrapped in a generic loop whose iterations preserve feasibility, since the loop remains valid with a random policy. Condition (ii) requires that the skeleton carry its own search logic, as branch-and-bound, large neighborhood search, dynamic programming, or Lin–Kernighan–Helsgaun do. A loop that without the learned policy reduces to a random walk does not satisfy it, and the mechanism belongs to M4. Three of the cases in Table 4 illustrate how the test is applied. For learned branching inside branch-and-bound, replacing the learned rule by pseudo-cost branching leaves a procedure that, for any admissible branching rule, terminates with a certified optimal solution and is an established OR method; both conditions hold, and the mechanism is M3. For learned local rewriting, replacing the learned policy leaves a loop that still terminates and returns a feasible solution, so condition (i) holds, but without the policy the loop reduces to random rewriting and is not an established method; condition (ii) fails, and the mechanism is M4. Heatmap-restricted dynamic programming shows why the mode and the inherited guarantee must be reported separately: removing the heatmap leaves full dynamic programming, which is valid and established, so the mechanism is M3, yet the deployed procedure searches a restricted state space and therefore inherits feasibility, termination, and exactness relative to that space rather than global optimality. The test fixes the mode; the guarantee that survives must be stated for the mechanism as deployed. Figure 4 contrasts the two control architectures.

Table 4. Classification of boundary cases with the procedure of Section 4.4.

Mechanism	Mode	Rationale
Learned branching in branch-and-bound (Gasse et al. [24])	M3	Default branching rules yield a valid, established solver (Q3a yes)
Neural diving: learned partial assignments completed by an MIP solver (Nair et al. [25])	M3	The solver completes and certifies the solution; the learned output acts as a warm start
Neural repair inside large neighborhood search (Hottung and Tierney [26])	M3	Large neighborhood search with a greedy repair is an established method
Learned candidate edges for LKH (Xin et al. [27])	M3	LKH with default candidates is state-of-the-art (Q3a yes)
Heatmap restricting dynamic programming (Kool et al. [28])	M3	Dynamic programming remains valid and established without the heatmap (Q3a yes); the restricted procedure retains feasibility, termination, and exactness over the restricted state space, but not completeness or global optimality
Learned local rewriting policy (Chen and Tian [29])	M4	The rewriting loop is valid but, without the policy, not an established method (fails condition ii)
Heatmap-guided Monte Carlo tree search for the TSP (Fu et al. [30])	Borderline M3/M4	Depends on whether the tree search is regarded as an established method independent of the heatmap
Learned optimizers by algorithm unrolling (Monga et al. [31]; Chen et al. [32])	Borderline M3/M4	M3 if unrolled iterations keep the classical convergence safeguards; otherwise M4
SPO+ training (Elmachtoub and Grigas [21])	M2	Decision loss enters training; the solver produces the decision
SPO Trees (Elmachtoub et al. [33])	M2	Decision-based evaluation guides tree construction without gradients
Weighted sample-average approximation (Bertsimas and Kallus [18])	M1	Weights are learned with statistical criteria, independently of the downstream decision
Constraint learning (Fajemisin et al. [8])	M1	A learned function is embedded in the model; it is trained with a statistical loss
Data-driven newsvendor decision rules (Ban and Rudin [34])	M4	The learned rule outputs the decision directly, trained on decision cost
Learning-augmented online algorithms (Lykouris and Vassilvitskii [35])	M3	Predictions are advice to a classical algorithm with robustness guarantees
LLM-generated heuristics (Romera-Paredes et al. [36]; Liu et al. [37])	M3	The generated program is the master algorithm; the LLM acts offline as a designer
LLM as a direct optimizer (Yang et al. [38])	M4	The generative model proposes solutions without an OR master algorithm
Agentic formulation and debugging (Ahmaditeshnizi et al. [4])	M5 (autonomous)	Acts on the modeling map; the numerical solution is produced by an MIP solver
Interactive preference construction (Lawless et al. [39])	M5 (assistive)	Acts on the human–model interface, Equation (17)

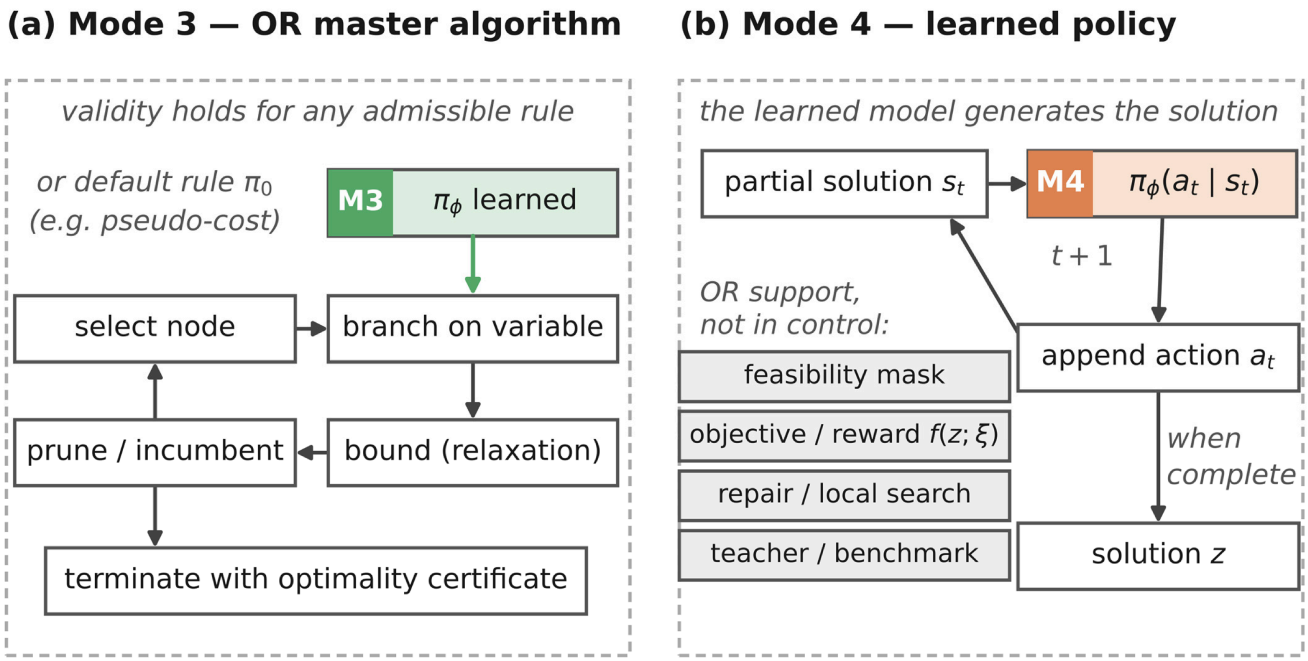


Figure 4. Control architecture of M3 and M4. In M3, a learned rule is plugged into an OR master algorithm whose validity does not depend on the rule; in M4, the learned policy generates the solution, and OR components provide support without governing the search.

4.5. Boundary Cases

Table 4 applies the procedure to mechanisms that are frequently classified inconsistently in the literature. Two cases remain genuinely borderline. For heatmap-guided tree search, the classification depends on whether the search component is regarded as an established OR method in its own right. For learned optimizers obtained by algorithm unrolling, the inherited property must be named precisely: the classical convergence result belongs to the procedure continued indefinitely with its original, non-learned parameters, whereas the deployed network has finite depth and learned parameters and inherits neither that convergence statement nor any optimality certificate. What may survive, when the unrolled iteration map keeps its iterates in the feasible set, is the feasibility of each iterate, and whether it does depends on the algorithm being unrolled. We report these cases rather than force them into one mode, because they mark where the functional vocabulary needs refinement.

5. Integration Modes in the Literature

This section reviews each mode, its methodological development, its main families, and representative works. The aim is not to enumerate every technique but to show how different research streams realize the mechanisms defined in Section 4. Figure 5 situates the representative works cited in this section on a common timeline.

5.1. Mode 1: Predict-Then-Optimize

5.1.1. Point-Estimate Pipelines

Predict-then-optimize is the most traditional and modular form of integration. A model estimates uncertain parameters such as demand, travel times, processing times, costs, or probabilities, and these estimates are used as inputs to an OR model. Modularity is its main strength: the predictor can be evaluated with conventional statistical metrics, the OR model keeps its structure and solution procedure, and either stage can be replaced without redesigning the other.

The implicit assumption is that better predictive accuracy leads to better decisions. The decision-focused learning literature has shown that this relation can be weak: prediction errors of similar magnitude can have very different decision consequences depending on the structure of the optimization problem [1,21]. Underestimating a parameter associated with an active constraint may change the optimal solution substantially, whereas a larger error in a region irrelevant to the decision may leave it unchanged. This limitation does not invalidate the paradigm. M1 remains attractive when predictive accuracy is a good proxy for decision quality, when predictors exist as independent components, or when modularity, interpretability, and independent validation of the predictive model are valued.

5.1.2. Beyond Point Estimates

M1 is not limited to point forecasts. Bertsimas and Kallus [18] show how machine-learning methods such as nearest neighbors, regression trees, and random forests can be used to construct weights over historical observations, which then define a conditional stochastic optimization problem of the form of Equation (4). The weights are learned with conventional statistical criteria, and the approach inherits consistency properties from the underlying estimators. Because the downstream problem does not enter the learning of the weights, it is classified as M1, although it is closely related to decision-focused approaches. Whether separate estimation or integrated estimation yields better decisions has a nuanced answer: Elmachtoub et al. [40] compare both from a stochastic-dominance perspective and show that integrated estimation–optimization is preferable when the model class is mis-specified, whereas estimate-then-optimize is preferable when the model class is well specified and data are sufficient.

A third form of M1 learns the uncertainty set of a robust optimization model. Johnstone and Cox [41] use conformal prediction to construct uncertainty sets with finite-sample coverage guarantees, and Goerigk and Kurtz [42] use deep neural networks to learn data-driven uncertainty sets that adapt to the shape of the data. In both cases, the learned object is a component of the model, and the robust counterpart is solved by standard OR methods.

Finally, learning may supply entire functions of the model. In optimization with constraint learning, a trained model, such as a regression tree, ensemble, or neural network, is embedded as a constraint or objective term of a mixed-integer model, typically because the relationship it represents has no known closed form [8,43]. Functionally, AI supplies a component of P and is trained with a statistical loss, so these mechanisms extend M1 from estimating parameters to estimating functions.

A related line of work, developed largely within the constraint programming community, learns the constraints themselves from examples of feasible and infeasible assignments rather than from a statistical fit to observed outcomes [44,45]. Although the terminology overlaps, the mechanism is the same in functional terms: AI supplies a component of the model that is then handed to a solver, which places both lines in M1.

5.2. Mode 2: Decision-Focused Learning

Decision-focused learning (DFL) integrates prediction and optimization during training, with the aim of improving downstream decisions. The predictor still estimates parameters used by an OR model, but the structure or result of the optimization explicitly influences how the predictor is trained [1]. Conceptual antecedents precede the term. Bengio [46] trained a model for financial decisions with a financial performance criterion rather than a purely predictive loss, and Kao et al. [47] proposed directed regression, which incorporates the decision objective into the fitting of linear models. The modern literature has consolidated around end-to-end frameworks that couple predictive models and optimization problems explicitly.

5.2.1. Differentiable Optimization Approaches

The first family builds a differentiable path from the prediction to the downstream decision, addressing the middle factor of Equation (7). OptNet [48] showed how quadratic programs can be embedded as differentiable layers in neural networks, obtaining gradients by implicit differentiation of the KKT conditions. Although OptNet is not a DFL framework in itself, it provided tools that later facilitated differentiation through optimization problems. Donti et al. [49] carried this idea to task-based end-to-end learning in stochastic optimization, training the predictor with a loss associated with the optimization task and obtaining gradients by differentiating the optimality conditions.

Wilder et al. [19] extended the approach to combinatorial problems. For problems whose relaxation is a linear program, a continuous relaxation alone does not solve the differentiability problem, because the optimal solution jumps between vertices. The authors therefore add a quadratic regularization term, which yields a strongly convex problem whose solution map is differentiable almost everywhere; at inference, the regularization can be removed to recover the original decision. The same group extended the approach to problems defined on graphs, where the graph or its attributes are only partially observed, combining learning and optimization for tasks such as community detection and facility location [50]. Vlastelica et al. [51] address black-box combinatorial solvers with linear objectives by constructing a backward pass from an interpolation of the loss, which requires only additional solver calls and avoids reimplementing the solver as a differentiable layer. Berthet et al. [20] perturb the inputs of the optimizer stochastically and consider the expectation of the optimal solutions (Equation (8)); the resulting operator is differentiable and not locally constant, its derivatives can be estimated by Monte Carlo sampling, and the framework connects with Fenchel–Young losses. Across this family, the common element is not a particular smoothing technique but the construction of a differentiation path through which decision quality modifies the predictor.

5.2.2. Surrogate Losses and Gradient-Free Approaches

A second family avoids differentiating through the argmin. Two situations should be distinguished: using a surrogate loss optimized with gradients or subgradients, and using genuinely gradient-free procedures. The smart predict-then-optimize framework of Elmachtoub and Grigas [21] belongs to the first group. It defines the SPO loss as the regret induced by a cost prediction and derives the convex surrogate SPO+ (Equation (9)). SPO+ is neither gradient-free nor solver-free: computing a subgradient requires solving the optimization problem with modified costs. What it avoids is differentiation of the argmin operator. The framework was soon extended beyond linear objectives: Mandi et al. [52] adapt SPO+ to hard combinatorial problems, showing that training on a continuous relaxation and warm-starting the solver makes the approach practical when each training iteration would otherwise require solving an NP-hard problem. In the second group, SPO trees [33] learn the structure of a decision tree by evaluating the quality of the decisions induced by candidate partitions; the optimization problem is still needed to evaluate those decisions, but training does not rely on backpropagation. Related surrogate formulations recast the problem as learning to rank feasible solutions rather than predicting coefficients accurately [53], while Shah et al. [54] replace hand-crafted surrogates with a loss function that is learned directly and is convex by construction, requiring only black-box access to the solver.

The distinction can be summarized by three independent questions: (i) does training differentiate through the solver; (ii) is the predictor trained with gradients or subgradients; and (iii) must the solver be invoked during training? SPO+ answers no, yes, and yes, respectively; other methods occupy different combinations.

5.2.3. Scope, Conditions of Usefulness, and Computational Cost

Most of the DFL literature assumes that uncertain parameters appear in the objective while the feasible set is known [1]. When unknown parameters enter the constraints, a wrong prediction can induce a decision that is infeasible under the true values, and minimizing objective regret alone is insufficient (Section 4.3, Equation (10)). A small but growing line of work addresses this setting. CombOptNet learns the constraints of integer programs [55]. Hu et al. [56] treat unknown constraint parameters in packing and covering linear programs as a two-stage process in which an infeasible predicted solution is corrected once the true parameters are revealed, and the cost of the correction enters the training loss. Mandi et al. [57] propose a single-stage alternative for generic constrained problems that avoids corrective actions: an infeasibility penalty loss penalizes predictions whose induced solution violates the true constraints, an optimality-preserving loss penalizes predictions that make the true optimal solution infeasible, and a weighted combination of the two lets the decision-maker trade infeasibility against suboptimality. Even so, the constraint setting remains much less mature than objective uncertainty, and methods such as SPO+ should not be extrapolated to it automatically.

DFL is not universally superior to M1. Its value appears mainly when the predictor is misspecified, has limited capacity, or is trained on insufficient data, so that some prediction error is unavoidable and should be placed where it matters least for the decision; Elmachtoub and Grigas [21] report particularly clear gains under misspecification. Empirical comparisons support this conditional view: Vanderschueren et al. [58] evaluate predict-then-optimize against predict-and-optimize across cost-sensitive learning tasks and find that neither dominates, with the advantage depending on the structure of the decision problem and the quality of the predictive model. Integrating the optimizer into training also has a computational cost, because many techniques require solving the downstream problem repeatedly. Mulamba et al. [59] address this bottleneck with contrastive losses and solution caching, replacing many solver calls by searches in a pool of previously computed solutions, which gives a controllable trade-off between training cost and the accuracy of the loss approximation. On the theoretical side, El Balghiti et al. [60] derive generalization bounds for the predict-then-optimize framework, relating the decision loss on unseen instances to the structure of the feasible region. On the practical side, PyEPO [61] provides an open library for end-to-end predict-then-optimize on linear and integer programs, which together with the benchmark of Mandi et al. [1] supports the comparability called for in Section 7.6.

5.3. Mode 3: AI-Enhanced Optimization

In M3, learning is integrated into the solution process to improve how an OR algorithm explores the solution space, makes internal decisions, or configures its search, while an OR master algorithm in the sense of Definition 1 retains global control. Three families act at increasing levels of abstraction: decisions during the run, guidance of the search, and configuration or design of the algorithm before the run.

5.3.1. Solver Decision Learning

Solver decision learning covers methods in which AI learns local or intermediate decisions of a solver. Learning to branch is the most consolidated example. Khalil et al. [62] showed that a ranking model can learn to imitate strong-branching decisions in mixed-integer programming, and Alvarez et al. [63] approximated strong-branching scores offline with ExtraTrees. Gasse et al. [24] consolidated this line with a graph convolutional network on the bipartite variable–constraint representation of an MILP, reducing feature engineering and showing generalization to larger instances. A parallel development concerns cut man-

agement, where AI does not generate new cuts but learns to select or prioritize valid ones: Tang et al. [64] formulated adaptive cut selection as a reinforcement-learning problem, and Paulus et al. [65] used imitation learning from lookahead experts. Nair et al. [25] combine learned branching with *neural diving*, in which a model proposes partial assignments that are completed and certified by the MIP solver. The same principle extends to node selection and primal heuristics: AI changes the trajectory of the solver, but the algorithmic structure that guarantees feasibility, search, and termination remains in place. The same lineage continues with hybrid architectures that reduce the inference cost of the learned rule at each node [66] and with representations that parameterize the state of the search tree rather than of individual candidates [67], while related work learns primal heuristics using graph neural networks to construct incumbent solutions [68]. Scavuzzo et al. [69] survey this line and its integration into full branch-and-bound implementations.

5.3.2. Learning-Guided Search and Heuristics

The second family covers approaches in which AI guides where or how an OR algorithm searches, identifying promising regions, building candidate sets, learning repair operators, or prioritizing heuristic components. Hottung and Tierney [26] incorporate a neural repair policy into large neighborhood search for the capacitated vehicle routing problem; large neighborhood search still destroys, repairs, accepts, or rejects solutions and organizes the iterations. In deep policy dynamic programming, Kool et al. [28] use a graph neural network to produce a heatmap of promising edges that restricts the state space explored by dynamic programming, while dynamic programming preserves states, feasible actions, dominance, and solution reconstruction. NeuroLKH [27] uses a sparse graph network to predict candidate edges and node penalties for the Lin–Kernighan–Helsgaun heuristic, which retains its local-search mechanism. Broader surveys classify many further uses of learning within metaheuristics, from initialization and operator selection to parameter control [9,10]. Applied instances of this idea appear in the OR literature: Karimi-Mamaghan et al. [70] integrate Q-learning into an iterated greedy algorithm to select operators during the search, and Kallestad et al. [71] propose a general deep reinforcement learning framework for selection hyper-heuristics. Across these works, a method does not belong to M4 merely because a neural network produces moves, scores, or candidates; by the substitution test, it belongs to M3 whenever the surrounding heuristic, metaheuristic, or dynamic program is an established method that governs the search.

5.3.3. Solver Configuration and Algorithm Design

The third family acts before the search. Automated algorithm selection learns which algorithm or configuration to use for an instance from its features; the per-instance selection literature shows that different algorithms dominate in different regions of the instance space, so that learning to select can outperform any single configuration [72]. Algorithm configuration extends this logic to solver parameters and components (Equation (12)). Hyper-heuristics also fit here: selection hyper-heuristics choose low-level heuristics during the search and are functionally close to learning-guided search, whereas generation hyper-heuristics construct new heuristics and are closer to algorithm design [73]. Recent work applies this logic directly to mathematical programming solvers: Ghaddar et al. [74] frame the choice of spatial branching rules as an algorithm selection problem, and Himmich et al. [75] develop an automatic tuner for MILP solvers.

LLMs have extended this family to the generation of algorithmic code. FunSearch pairs an LLM with an evaluator in an evolutionary loop to discover programs, including heuristics for combinatorial problems [36]; Evolution of Heuristics [37] and ReEvo [76] evolve heuristic code and its natural-language description with LLMs acting as hyper-

heuristics. The same approach has been applied to mixed-integer programming, where LLMs generate heuristics for specific problem classes [77]. Using a generative model does not change the functional classification: when the LLM acts offline as a designer and the generated program is itself an OR master algorithm in the sense of Definition 1 at run time, the mechanism belongs to M3; code generation alone is not sufficient, since the generated procedure must govern the search and carry validity properties of its own. Warm starts belong to this family when a learned output initializes a classical solver; if, instead, the learned model produces the final solution and no master algorithm governs the process, the mechanism belongs to M4.

5.3.4. Learning-Augmented Algorithms as a Cross-Cutting Paradigm

Learning-augmented algorithms, also known as algorithms with predictions, form a theoretical stream that cuts across M3. They incorporate learned predictions as advice to a classical algorithm, aiming to perform better when predictions are informative without giving up worst-case guarantees when they are wrong. Lykouris and Vassilvitskii [35] introduced this approach for competitive caching, and Mitzenmacher and Vassilvitskii [14] consolidated it as a general framework, which analyzes consistency, robustness, and, in later work, smoothness, the rate at which performance degrades as prediction error grows (Equation (13)). The stream matters because it shifts attention from average improvement to the guarantees that remain when AI fails, a theme that recurs across all modes (Section 7.1). The paradigm has been developed for problems that are familiar in OR, including online matching [78] and scheduling with imperfect predictions [79], and it has been connected to classical algorithmic tools such as the primal–dual method [80].

5.4. Mode 4: Learning-Based Optimization

In M4, the learned model is the principal mechanism that generates or improves solutions, although it may use OR structures to enforce feasibility, evaluate solutions, or complement the procedure. The dominant literature is that of neural combinatorial optimization (NCO), but the functional category is broader: NCO describes a technological family, whereas M4 is defined by the role of the learned model. Amortized optimization, decision rules, and generative models used as direct optimizers belong to the same mode.

5.4.1. Solution Construction

Constructive methods build a solution sequentially, each partial decision conditioning the next. Pointer networks [81] replaced a fixed output vocabulary with an attention mechanism that points to input positions, which allowed outputs whose size depends on the instance and yielded approximate solutions to geometric problems including the traveling salesman problem (TSP); supervised training, however, required target solutions. Bello et al. [82] replaced labels with reinforcement learning, training a recurrent policy over permutations with the tour length as reward. Conceptually, the learned model no longer predicts parameters for a solver or decides an internal action of a classical algorithm: it constructs the solution itself. Kool et al. [83] consolidated this line with an attention-based encoder–decoder trained with REINFORCE (Equation (15)) and applied it to the TSP, the capacitated vehicle routing problem, the orienteering problem, and the prize-collecting TSP. POMO [84] exploits the existence of multiple equivalent representations of a solution by launching rollouts from different starting points and using a shared low-variance baseline.

Sequential construction usually relies on feasibility masking to prevent actions that violate simple constraints. Masking becomes problematic when deciding whether a partial action can still be completed feasibly is itself difficult. Bi et al. [85] address this limit with proactive infeasibility prevention, which adds Lagrangian-based information and

preventive masks, and a variant that learns to predict such masks with an auxiliary decoder. The work illustrates a shift from benchmarks with simple constraints towards neural solvers for more complex feasibility structures.

5.4.2. Solution Improvement and Refinement

Improvement methods start from a solution and learn a policy to modify it iteratively. NeuRewriter [29] learns, with actor–critic reinforcement learning, a factorized policy that first selects a region of the solution and then a local rewriting rule, and it was applied to expression simplification, online job scheduling, and vehicle routing. Wu et al. [86] learn improvement heuristics for routing by selecting pairwise local operators with a self-attention policy. Improvement solvers offer a natural trade-off between time and quality, at the price of depending on the initial solution, the learned neighborhood, and the ability to escape local optima or infeasible regions. The boundary between construction and improvement need not be rigid: construct-and-refine [87] trains a constructive module to produce diverse, high-quality solutions suited to light refinement, with both components sharing representations, to reach feasibility and quality with few refinement steps.

5.4.3. One-Shot, Amortized, and Decision-Rule Approaches

A third group maps an instance or context to a decision in a single evaluation. Amortized optimization learns a model that approximates the solution map of a family of problems, trading offline training for fast inference [88]. For continuous problems with hard constraints, DC3 [89] combines a learned predictor with completion and correction steps that restore feasibility. On the contextual side, Ban and Rudin [34] learn newsvendor decisions directly from features by empirical risk minimization of the decision cost; such decision rules replace the optimization step entirely and therefore belong to M4 under the authority criterion (Section 2). LLMs have also been used as direct optimizers that iteratively propose candidate solutions from a history of evaluated ones [38]. Learned optimizers obtained by unrolling iterative algorithms [31,32] lie on the M3/M4 boundary discussed in Section 4.5.

5.4.4. Cross-Cutting Challenges: Generalization, Scalability, and Feasibility

Good performance on the training distribution does not ensure good performance on larger or differently distributed instances. Joshi et al. [90] show for the TSP that models performing well on small instances can degrade substantially when extrapolated to practical sizes; generalization should therefore be assessed across size, distribution, and structure, not only on an i.i.d. test set. Subsequent work addresses these limitations directly: BQ-NCO reformulates the problem to improve out-of-distribution generalization [91], and other approaches target real-time solution of large-scale routing instances [92]. Systematic evaluations against classical solvers remain scarce, but Liu et al. [93] provide one for the traveling salesman problem. Scalability is related: training can be expensive, attention-based architectures scale poorly with instance size, and fast inference does not imply a global advantage if large training volumes are needed (Section 6.2). Feasibility is the third challenge; the works of Bi et al. [85,87] show that constraint handling is becoming a line of development in its own right. These aspects are treated as cross-cutting challenges rather than subfamilies because they affect both construction and improvement methods. Recent tutorials characterize NCO as a family of learned heuristics and identify generalization, scalability, and realistic constraints as open issues [2].

5.5. Mode 5: AI-Augmented OR Modeling and Decision Support

In M5, AI acts on the interface between the real-world problem, its formal representation, the OR model, and the decision-maker. Its function may be to formulate models, assess their correctness, translate preferences and queries, or explain the consequences of changes; numerical optimization normally remains with OR methods. LLMs have driven recent growth because they offer a natural interface between human language and formal representations, but the mode is not defined by LLMs, and earlier non-generative work realizes the same functions (Section 5.5.3).

5.5.1. Optimization Modeling

The first family seeks to reduce a long-standing barrier to OR adoption: transforming a problem description into a correct formulation and then into executable code. Ramamonjison et al. [3] are a central antecedent of autoformulation: their system receives textual descriptions of linear programs and generates formulation suggestions that the user can validate and edit, framing the task as augmented intelligence for OR modeling. The NL4Opt competition formalized the problem as a reproducible benchmark, separating semantic entity recognition from the generation of a meaning representation convertible into a solver-ready formulation [94]; its contribution is primarily methodological, providing common data, tasks, and metrics. Multi-agent decompositions emerged as the dominant strategy for complex problems: Chain-of-Experts assigns specialized roles to several LLMs that cooperate on the formulation [95], and related frameworks couple modeling with an explicit verification stage [96]. Fine-tuning smaller open models for the formulation task has also been explored [97]. OptiMUS extends modeling to an agentic workflow that decomposes complex problems, develops mathematical models, writes and executes solver code for (MI)LP, evaluates solutions, and corrects modeling or implementation errors iteratively [4]. In parallel, Huang et al. [98] propose ORLM, a customizable framework for training large models for automated optimization modeling. The progression from suggestion systems to agents and specialized models also marks the move from assistive to autonomous M5.

5.5.2. Model Verification

Producing a formulation that is syntactically valid, or code that runs, does not show that the model represents the intended problem. Model verification comprises methods that assess whether a generated formulation is correct, consistent, or equivalent to another representation. Astorga et al. [99] make formulation correctness an explicit component of autoformulation, using LLMs within Monte Carlo tree search both to generate formulation hypotheses and to evaluate them, with pruning of trivially equivalent formulations. Solver-informed reinforcement learning [100] uses verifiable signals derived from executable code and instance files to reward syntactic validity, feasibility, and solution quality when training LLMs for modeling; agreement in feasibility or objective value, however, is not by itself a proof of semantic equivalence. EquivaMap [22] addresses equivalence directly: it defines Quasi-Karp equivalence through the existence of a mapping between decision variables, uses an LLM to propose the mapping, and verifies on solved instances that mapped solutions preserve feasibility and optimality, an instance-level application of the criterion in Equation (16). Instance-level checks, however, can miss inconsistencies that appear only on other instances: Robbins et al. [23] report LLM-proposed cutting planes that pass instance-level validation yet remove optimal solutions for some instances, and propose FLARE, which combines an LLM-based agent with the Lean proof assistant to verify MILP reformulations at the formulation level and to issue a machine-checkable certificate for every reformulation it accepts (a preprint

at the time of writing). Benchmarks complement these methods by measuring modeling performance systematically rather than certifying a single model: OptiBench evaluates LLMs on optimization modeling tasks and proposes a data-synthesis method to improve them [101]. Verification methods thus range from correctness-aware search and solver-grounded feedback to instance-level equivalence checking and formal, formulation-level certification, each verifying specific properties under specific assumptions.

5.5.3. Interactive Decision Support

The third family concerns the interaction between the OR system and the decision-maker, in both directions. From the human to the model, AI helps express preferences, constraints, queries, and scenarios; from the model to the human, it turns results, trade-offs, and sensitivities into understandable explanations. Lawless et al. [39] study this function with MeetMate, an interactive system for meeting scheduling in which an LLM helps interpret and progressively develop preferences expressed in natural language and integrates them into a constraint-based representation, an instance of Equation (17). Preferences are not assumed to be fixed before the interaction; they emerge and change during the dialogue. The prototype works with a small set of alternatives and weighted constraints, so its main contribution is the iterative interface between LLM and constraints rather than integration with a large-scale solver. OptiChat [102] targets the inverse direction, allowing practitioners without optimization expertise to interrogate an existing model and understand infeasibility or unexpected results. Explainable operational research [103] addresses the reverse direction: it introduces decision information obtained through what-if analysis, represents the OR model as a bipartite graph, uses graph edit distance as a structural proxy for the magnitude of model changes, and uses LLMs to turn the quantitative analysis into explanations.

These mechanisms differ in whether preference information is mediated by a learned component or handled by a fixed elicitation protocol, and only the former belongs to M5: interactive procedures that elicit and apply preferences without any learned component realize the same function and are better read as precursors. The functions of M5 are not tied to generative models. Battiti and Passerini [104] learn a model of the decision-maker's utility from pairwise judgments of candidate solutions, using support vector ranking, and use it to steer an evolutionary multi-objective search, an early learned instance of Equation (17); related work learns value functions from preference information elicited during interactive multi-objective search [105]. Other work serves the same functions without learned components and is better regarded as a precursor than as an instance of M5: interactive multi-objective optimization has long studied how preference information elicited during the search guides the computation of solutions [106], and counterfactual explanations of optimization-based decisions can be computed through inverse constraint programming [107]. Forel et al. [108] address explanation for pipelines that do contain learned components, deriving counterfactual explanations of decisions produced by data-driven optimization. Together, these works support treating M5 as a functional category defined by the role of a learned component at the modeling and interaction layer, not by the use of LLMs.

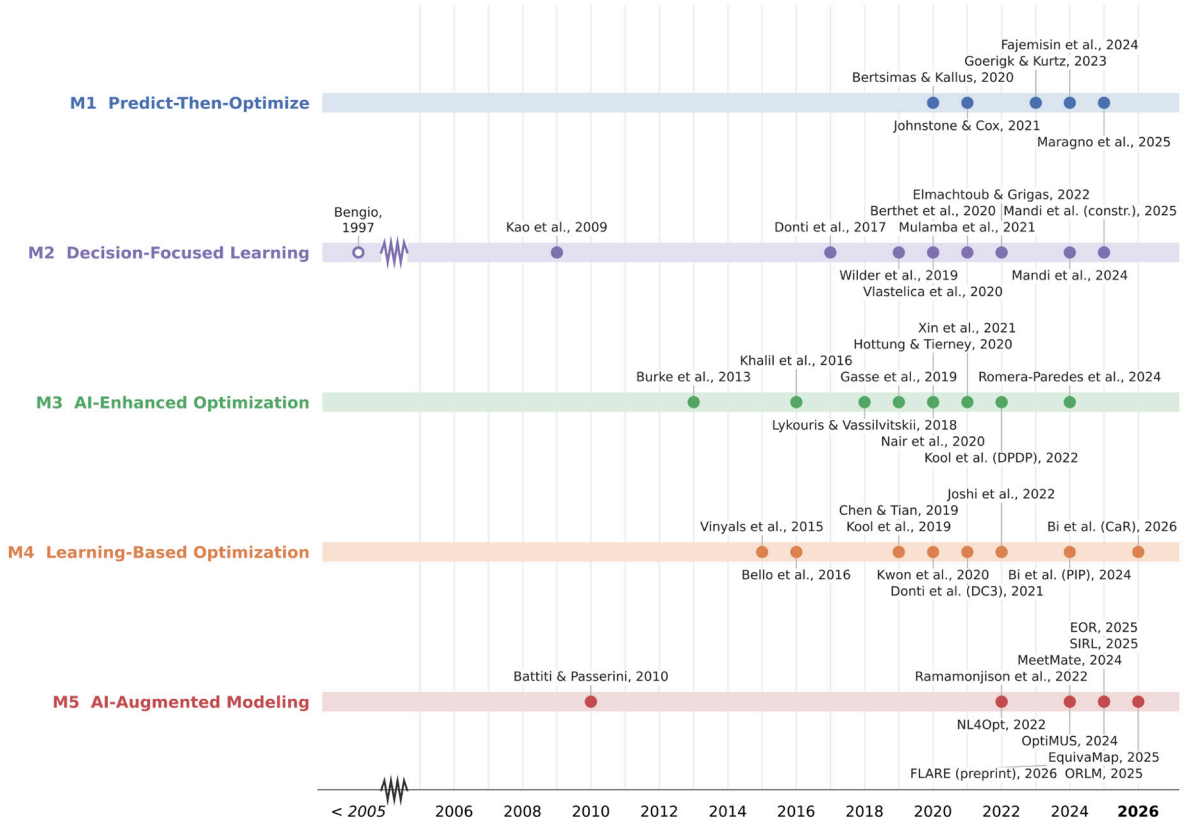


Figure 5. Representative works cited in Section 5, by integration mode and year of publication. Works are labelled by author and year; the corresponding reference numbers are [1,3,4,8,18–29,35,36,39,41–43,46,47,49,51,57,59,62,73,81–85,87,89,90,94,98,100,103,104]. The time axis is broken at 2005 (zigzag): the only earlier work, Bengio [46], is shown to the left of the break with an open marker, so that the axis is not stretched by a single early reference. The figure is illustrative and does not represent a bibliometric count.

6. Cross-Mode Analysis

The five modes differ not only in the AI technique employed but in where learning enters the decision process, how much control OR retains, and what kind of uncertainty or complexity AI is meant to manage. This section compares them along the dimensions of Section 4.2 and analyzes how they interact.

6.1. OR as a Source of Structure

The modes differ in how deeply they rely on explicit OR structure. M1 can be fully modular. M2 requires the structure of the optimization problem during training. M3 depends directly on solver mechanics, since the learned component must interact with branching, cutting, neighborhoods, or dynamic programming states. M4 may use less explicit structure at inference, but OR knowledge does not disappear: feasibility masks, objective functions, problem symmetries, classical heuristics, and benchmark solvers provide inductive bias, training signals, evaluation criteria, and repair mechanisms. M5 relies on mathematical semantics, solver feedback, equivalence criteria, and model sensitivities to constrain and verify generative AI. Across the framework, OR acts as solver, teacher, differentiable layer, constraint system, verifier, benchmark, or semantic grounding, and not only as an alternative to AI.

6.2. Training Signal, Data, and Computational Burden

M1 relies on conventional labeled data. M2 requires the same data plus access to the downstream problem during training, often at substantial computational cost. M3 may require expert labels from expensive solver decisions, reinforcement-learning interaction with a solver, or large collections of solved instances. M4 frequently depends on repeated simulation or optimization episodes and substantial offline training before offering fast inference. M5 needs paired representations of descriptions, formulations, code, solver outcomes, explanations, or interactions, and its main data problem is semantic correctness rather than size: a large corpus of executable models may still contain incorrect formulations. Xiao et al. [15] found a surprisingly high error rate in widely used benchmark datasets for LLM-based modeling and released cleaned versions with a revised leaderboard.

Computational efficiency must therefore be assessed end to end. For a learned solver that replaces a classical method of comparable solution quality, the number of instances needed before the learned approach pays off is

$$N^{\dagger} = \frac{C_{\text{data}} + C_{\text{train}}}{\bar{c}_{\text{OR}} - \bar{c}_{\text{inf}}}, \qquad \bar{c}_{\text{OR}} > \bar{c}_{\text{inf}}, \tag{18}$$

where C_{data} and C_{train} are the one-off costs of generating training data and training, and $\bar{c}_{\text{OR}}$ and $\bar{c}_{\text{inf}}$ are the average per-instance costs of the classical method and of learned inference, respectively. Equation (18) makes explicit what is often left implicit in reported speed-ups: they are meaningful for repetitive applications with many similar instances and much less so for one-off problems. When solution quality differs, the comparison must be quality-adjusted, and the same reasoning applies to the training overhead of M2 and to the feature-extraction and inference overhead of M3.

6.3. Guarantees, Robustness, and Failure Modes

A central trade-off concerns the guarantees inherited from OR and the failure modes introduced by learning (Table 5). M1 and M2 usually retain the feasibility and optimality properties of the downstream solver for the predicted instance, but they are exposed to model error: the decision may be poor, or infeasible if parameters enter the constraints, under the true parameters. M3 preserves validity by Definition 1 (i), while learned branching, cut selection, or search guidance may affect run time and practical performance in ways that are difficult to predict. M4 faces the strongest tension between speed and classical guarantees (Section 5.4.4). M5 introduces a different class of risk: semantic errors in a generated formulation or explanation may remain hidden even when the numerical solver behaves perfectly. Across the five modes, performance under accurate predictions is not a sufficient evaluation criterion.

Table 5. Cross-mode comparison of AI–OR integration mechanisms.

Mode	Primary Locus of AI	Decision Authority	Role of OR	Typical Training Signal	Principal Risk
M1 Predict-Then-Optimize	Before optimization	OR solver	Downstream decision model	Predictive loss	Prediction error misaligned with decision quality
M2 Decision-Focused Learning	Predictor training	OR solver at inference	Shapes learning through the downstream problem	Decision-aware loss	Training cost; scope assumptions; feasibility under uncertain constraints
M3 AI-Enhanced Optimization	Inside or before the solver	OR master algorithm	Controls search and validity	Expert decisions, RL reward, performance data	Generalization of learned behavior; overhead

Table 5. *Cont.*

Mode	Primary Locus of AI	Decision Authority	Role of OR	Typical Training Signal	Principal Risk
M4 Learning-Based Optimization	Solution generation or improvement	Learned model or policy	Masks, rewards, repair, teachers, benchmarks	Solution quality (supervised or RL)	Out-of-distribution behavior, feasibility, scalability, weak guarantees
M5 AI-Augmented Modeling and Decision Support	Representation and human interface	Usually OR for optimization; human remains central	Formal model, solver grounding, verification	Language–model pairs, solver feedback, interaction data	Semantic errors, hallucination, accountability

6.4. Human Role and Accountability

In M1–M4, the human is usually outside the computational loop once the model, objective, and constraints have been specified, and interpretability concerns focus on predictive features, learned solver decisions, or learned policies. M5 reintroduces the human into the integration mechanism through formulation assistance, preference elicitation, what-if analysis, verification, and explanation. This matters in high-stakes or weakly specified problems, where preferences evolve, stakeholders disagree, and assumptions must be negotiated; in such contexts, AI may add more value by improving the interaction between humans and formal models than by replacing the optimizer. Accountability shifts accordingly: when AI predicts an input, model risk lies in the predictive layer; when it constructs a solution, responsibility extends to the learned policy; when it formulates or explains a model, errors can alter the meaning of the decision problem itself.

The distinction between assistive and autonomous M5 (Table 2) has practical consequences. In assistive systems, a human validates formulations or preference updates and remains accountable for the model; errors are detectable at the interface. In autonomous systems, an agent formulates, solves, and debugs with limited human validation, which raises throughput but moves the burden of correctness onto automated verification. The recent evolution of M5 suggests that one of the most active frontiers of AI–OR is moving from the automation of isolated algorithms towards systems that close the loop between problem description, formalization, validation, solution, and interaction, which increases the importance of correctness, provenance, and human oversight (Section 7.4).

6.5. Interactions Between Modes and Composite Systems

Modes interact in systematic ways (Figure 6). A shared predictor can move from M1 to M2 by changing only its training objective. Predicted inputs instantiate the instances solved by enhanced or learned solvers. Enhanced or learned solvers can reduce the cost of the solver calls inside M2 training, which is the main computational bottleneck of that mode. M3 and M4 exchange information in both directions: learned solutions provide warm starts, neighborhoods, or candidate sets to OR algorithms, and OR algorithms act as teachers, repair operators, and benchmarks for learned ones. M5 defines the task of every other mode through the formulation, and receives their solutions, sensitivities, and prediction uncertainty as material for explanation and what-if analysis.

A composite system can be written as a composition of mechanisms, for instance

$$\hat{z} = A_{\pi_\phi}(\xi), \qquad \xi = \left(M_\varphi(d),\, m_w(x)\right), \tag{19}$$

in which a formulation produced by M5, parameters produced by M1 or M2, and a solver enhanced by M3 jointly determine the decision. If $\tilde{z} = z^*\left(\hat{\theta}\right)$ denotes the exact optimum of the formulated model with predicted parameters, and $\hat{z}$ is feasible for the true problem, the loss of a composite system decomposes as

$$f(\hat{z};\theta) - f(z^*(\theta);\theta) = \underbrace{f(\hat{z};\theta) - f(\tilde{z};\theta)}_{\text{solution error}} + \underbrace{f(\tilde{z};\theta) - f(z^*(\theta);\theta)}_{\text{estimation and modeling error}}. \tag{20}$$

Equation (20) is an identity, but it clarifies attribution: the two terms can have opposite signs, so improving one component in isolation does not guarantee a better system, and evaluating components separately can be misleading. When the generated model is formulated over different decision variables, the solution must first be carried into the intended decision space through the map of Equation (16) before the true objective can be evaluated, and the decomposition is read with the mapped solution in place of the original. Figure 7 illustrates a composite system for Example 1 in which all five modes appear.

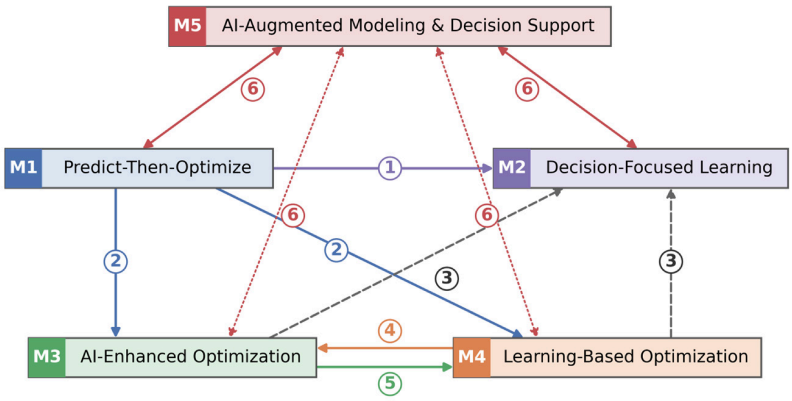


Figure 6. Interactions between integration modes in composite AI–OR systems. *The numbered arrows correspond to the interactions listed in the key below the diagram; double-headed arrows denote exchanges in both directions.*

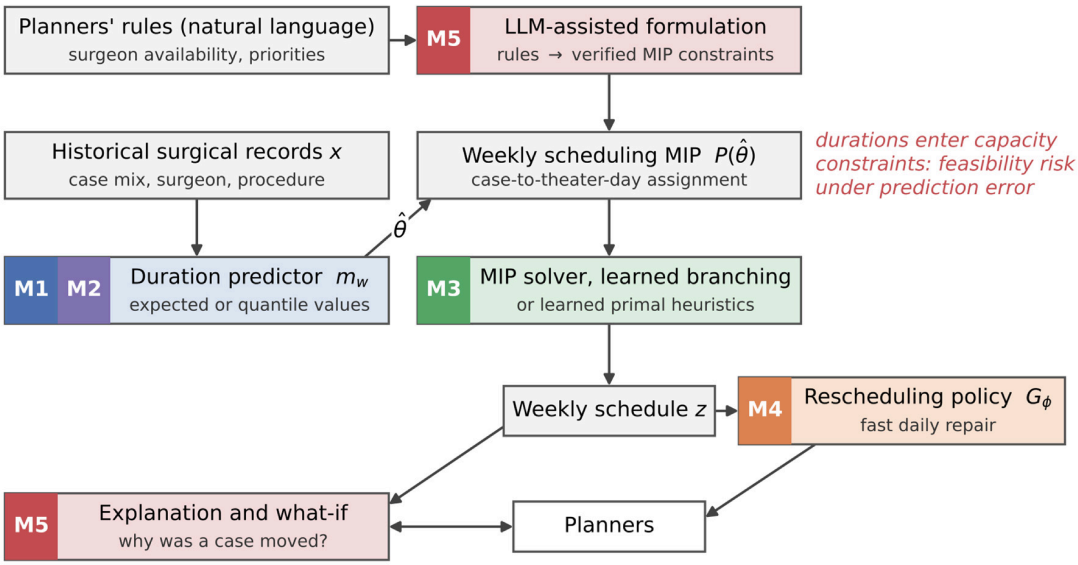


Figure 7. A hypothetical composite system for weekly surgical scheduling (Example 1) in which each component corresponds to one integration mode. The system is illustrative and does not describe an implemented application. Arrows show the flow of information between components; the coloured tab on each box identifies the integration mode that the component realizes.

Taken together, the analysis indicates that AI–OR integration is not a continuum from prediction to autonomous optimization but a set of distinct ways of reallocating information, computation, decision authority, and interaction across the decision process. This perspective provides the basis for identifying research gaps both within modes and at their interfaces.

7. Research Gaps and Future Directions

The framework shows that many consequential gaps arise at the interfaces between learning, optimization, formal modeling, and human decision-making, and become visible only when the literature is examined across modes. Six directions are identified below, each linked to the formal framework of Section 4.

7.1. Reliability and Generalization Under Model Error

Learned components are commonly evaluated under conditions similar to those of training, yet each mode introduces its own pathway from model error to decision error: distorted inputs in M1, misweighted trade-offs in M2, poor guidance in M3, poor or infeasible solutions in M4, and semantically wrong formulations or explanations in M5. Future work should move from average performance to explicit degradation analysis. For a mechanism whose learned component has error level η, a *degradation profile*

$$\Delta(\eta) = \mathbb{E}\left[f\left(\hat{z}_\eta; \theta\right)\right] - \mathbb{E}[f(\hat{z}_0; \theta)] \tag{21}$$

describes how decision quality deteriorates as the component becomes less accurate, less calibrated, or increasingly out of distribution. Learning-augmented algorithms already bound such profiles through consistency and robustness (Equation (13)), but comparable results are scarce for the other modes. A general design principle follows: learned components should expose their uncertainty and support safe reversion to classical OR mechanisms when confidence is insufficient.

Generalization remains a major challenge, particularly in M3 and M4, where learned components are often trained on synthetic instance families with fixed distributions, structures, or sizes. At least four forms should be distinguished: larger instances, shifted parameter distributions, new structural patterns, and transfer across problem variants. Benchmarks should include controlled distribution shifts and extrapolation regimes that test whether learned behavior reflects transferable structure or regularities of the training generator. Hybrids that use learning to accelerate common cases while relying on OR structure to preserve feasibility or recover performance outside the learned regime, typically M3 mechanisms with fallback to π_0, offer a realistic path to deployment where instance distributions are unstable.

7.2. Decision-Focused Learning with Uncertain Constraints

The DFL literature is considerably more mature when uncertainty affects objective coefficients than when it affects constraints (Section 5.2.3). In many operational problems, uncertain processing times, capacities, or service durations determine feasibility rather than cost, as in Equation (2). When such parameters are predicted, decision quality cannot be evaluated by regret alone; the learning objective must reflect the probability, severity, or cost of violations, as in Equation (10). Recent work has begun to address this setting, either through a second-stage correction whose cost enters training [56] or through single-stage losses that balance infeasibility against suboptimality with an explicit weight [57]. These contributions sharpen rather than close the agenda, and five directions stand out.

First, in the approaches reviewed here, training still requires solving the constrained problem for every instance at each iteration; adapting the solution-caching ideas devel-

oped for objective uncertainty [59] is a natural next step, as Mandi et al. [57] themselves note. Second, joint prediction of objective and constraint parameters, which is the common case in practice, is identified as future work in that line. Third, evaluation requires care: regret computed only over instances whose predicted solution is feasible can be misleading when many solutions are infeasible, so feasibility rates and regret should be reported together (Table 6). Fourth, the weight that trades infeasibility against suboptimality expresses the decision-maker's risk attitude rather than a purely technical hyperparameter. In Example 1, it encodes how planners weigh overtime and cancellations caused by under-predicted durations against idle theater time caused by conservative schedules. Eliciting this weight interactively would connect M2 with the preference mechanisms of M5 (Equation (17)), an interface we did not find addressed in the reviewed literature. Fifth, connections with chance constraints, robust and distributionally robust optimization, and recourse models remain promising routes to extend M2 from cost-aware to constraint-aware learning. The same issue links M2 and M4: learned predictors and learned solvers both need mechanisms that distinguish a numerically attractive solution from one that is operationally admissible.

Table 6. Minimum reporting items for end-to-end evaluation of AI–OR mechanisms.

Aspect	What to Report
Decision quality	Objective value or regret under the true parameters; when parameters enter constraints, feasibility rate, and violation magnitude reported jointly with regret, since regret restricted to feasible instances can mislead
Computational cost	Training, data-generation, and inference cost; break-even horizon, Equation (18); solver versions, hardware, time limits, and tolerances
Robustness	Performance under controlled distribution shift and size extrapolation; degradation profile, Equation (21)
Human-facing output	Semantic error categories for formulations; validity, not only fluency, of explanations
Reproducibility	Instance generators, random seeds, training budgets, and whether reported times include preprocessing and inference overhead

7.3. Compositional Systems

Most studies evaluate one mechanism in isolation, yet realistic systems combine several (Equation (19)). Their behavior cannot be understood by evaluating each component independently, because errors can compound or offset (Equation (20)). Open questions include how uncertainty propagates across components, how responsibility should be allocated between modules, and when one mode should trigger or override another. This motivates *adaptive mode selection*: rather than fixing an architecture, a system could decide per instance whether to trust a prediction, invoke a classical solver, use a learned heuristic, request human clarification, or fall back to a verified formulation, depending on instance features and estimated uncertainty. The interaction map in Figure 6 can serve as a checklist of interfaces that such systems must manage.

7.4. Verification, Semantic Correctness, and Provenance

M5 exposes a gap that grows as generative AI enters OR workflows: syntactic validity and solver success are weak proxies for semantic correctness. Layered verification architectures should combine language-level consistency checks, symbolic reasoning, solver-based

validation, instance-level equivalence testing in the sense of Equation (16), formulation-level certification with proof assistants [23], generation of test instances, and human review, addressing not only whether a formulation executes but whether variables, constraints, units, objectives, and assumptions preserve the intended meaning. The same caution extends to generated explanations: evidence from natural language processing shows that the mechanism producing a rationale can itself be manipulated, so fluency is not evidence that an explanation reflects the model it purports to describe [109]. Provenance is a related need: generated formulations and explanations should keep traceable links to source statements, assumptions, model transformations, and solver outputs, to support debugging, auditing, and accountability.

7.5. Human-Centered and Interactive Optimization

Much of the AI–OR literature assumes that objectives and constraints are fully specified before optimization. Real decision-making is iterative: stakeholders revise preferences after seeing alternatives, discover missing constraints, and negotiate trade-offs. Interaction should be treated as part of the optimization process (Equation (17)), rather than as an interface added afterwards. This includes preference elicitation, conversational what-if analysis, explanation of assumptions, interactive constraint refinement, and detection of situations in which the system lacks information to proceed safely. Explanations must remain grounded in model structure and solver output rather than in fluent but unsupported text; combining generative interfaces with sensitivity analysis, counterfactual optimization, or verified model transformations is a promising route.

7.6. Evaluation Beyond Isolated Benchmarks

Evaluation practices are fragmented: predictive systems report statistical error, solver-learning studies report run time or gaps, DFL reports regret, neural solvers report solution quality and inference time, and LLM-based modeling reports formulation or code accuracy. These metrics are useful but insufficient for comparing complete systems. End-to-end protocols should report, at a minimum, the items in Table 6.

8. Discussion

The literature reviewed shows that the relationship between AI and OR cannot be described by a single methodological trajectory. AI does not simply replace OR, nor does OR merely provide a downstream layer for predictive models. Learning can estimate inputs, shape predictor training through optimization, modify solver behavior, construct solutions, generate and verify models, or mediate interaction with decision-makers. Technology-centered classifications remain useful within communities but become difficult to apply as the same technologies migrate across the pipeline. By shifting the unit of classification from the AI technology to the integration mechanism, and by defining mechanisms formally, the framework provides a criterion that remains stable as architectures change, and it makes explicit where it agrees with and departs from existing taxonomies (Section 2).

Viewed historically, the literature moves from loosely coupled pipelines towards increasingly integrated systems, and M5 extends integration beyond the numerical algorithm to the whole decision lifecycle. This evolution should not be read as a replacement sequence in which tighter coupling is superior. M1 may be preferable when modularity, transparency, maintainability, or independent validation of the predictive model are important. Classical solvers may be preferable to learned ones when exactness, worst-case guarantees, or reliability under distribution shift are required, and when Equation (18) does not favor learning. The relevant question is not which mode will dominate but which mechanism is

appropriate for a given decision context. Across the modes, OR provides structure (objectives, constraints, search algorithms, optimality conditions, sensitivities, and verification criteria), while AI provides adaptation, approximation, representation learning, and natural interfaces; the strongest systems are likely to assign each component the role for which it has the greatest comparative advantage.

Limitations

This framework has several limitations. First, boundaries can become ambiguous in highly hybrid methods. The substitution test resolves many cases, but some, such as heatmap-guided tree search and unrolled optimizers, remain borderline (Table 4), and the second condition of Definition 1 ("established OR method") involves judgment. Second, the maturity of the literature is uneven: M1–M4 have established traditions, whereas LLM-based modeling, verification, and interaction evolve rapidly, so the internal structure of M5 is the most likely to change. Third, the review reflects the terminology visible in the current literature; adjacent fields such as automated reasoning, constraint programming, data-driven control, and human–AI decision support sometimes describe related mechanisms with different vocabulary. Fourth, the taxonomy is conceptual and tested against coded studies rather than derived from bibliometric clustering; its value should be judged by its explanatory usefulness, its ability to classify boundary cases consistently, and its capacity to reveal gaps. Larger coding exercises with independent coders would provide a natural test of its reliability.

9. Conclusions

AI and OR are increasingly integrated across the decision-making process, yet the literature remains fragmented across communities that use different terminology for closely related mechanisms. This review has proposed a functional taxonomy that classifies AI–OR integration by the role AI plays relative to the OR process rather than by the learning architecture, problem class, or solver technology.

Five modes were identified and defined formally. Predict-then-optimize uses AI to estimate inputs or components of an OR model with a statistical loss. Decision-focused learning lets the downstream optimization problem shape predictor training. AI-enhanced optimization learns rules, configurations, or designs for an OR master algorithm whose validity does not depend on them. Learning-based optimization assigns the learned model the principal role in generating or improving solutions. AI-augmented OR modeling and decision support extends integration to formulation, verification, preference elicitation, and explanation. A decision tree and a substitution test turn these definitions into an explicit and inspectable classification procedure.

The gaps identified suggest that the next stage of AI–OR research may be less about replacing one component with a learned alternative and more about building reliable decision systems in which prediction, optimization, learned search, generative modeling, verification, and human interaction are coordinated explicitly. Three principles emerge. First, functional role should guide method design and evaluation: the same technology requires different safeguards depending on whether it predicts parameters, guides a solver, produces a solution, or formulates a model. Second, OR structure should be treated as a source of inductive bias and verification, not merely as a competing baseline. Third, uncertainty about the learned component itself should become a first-class element of the decision process.

The modes are not mutually exclusive categories of applications but mechanisms that coexist and interact within composite systems, and the cross-mode analysis shows that OR remains central in all of them, as decision model, master algorithm, differentiable

structure, source of feasibility and training signals, or formal grounding for generated models. Several challenges cut across the framework: generalization beyond benchmark distributions, feasibility and robustness under model error, particularly when uncertain parameters enter constraints, the computational cost of integrated training, the semantic correctness of generated formulations, reliable explanation, and the evaluation of complete pipelines. Future progress is likely to depend less on stronger predictive or generative models alone than on architectures that coordinate learning, optimization, verification, and human judgment by design. The framework is offered as a common vocabulary for that effort, to be refined as the field evolves, while retaining its central principle: AI–OR integration is best understood by asking what role AI performs in relation to the decision and optimization process.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/math14224192/s1, Table S1: Coding matrix of the studies included in the review, reporting for each reference the assigned integration mode, the values of the five classification dimensions of Section 4.2, the subsection in which it is reviewed, and the sections in which it is cited.

Author Contributions: Conceptualization, F.B. and R.C.; methodology, F.B., R.C., and E.F.; formal analysis, F.B. and R.C.; investigation, F.B. and R.C.; writing—original draft preparation, F.B.; writing—review and editing, R.C., E.F., O.C., and R.R.; visualization, F.B. and R.C.; validation, E.F., O.C., and R.R.; supervision, E.F.; project administration, F.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article and in the Supplementary Material. Further inquiries can be directed to the corresponding author.

Acknowledgments: During the preparation of this manuscript, the authors used Claude Opus 5 (Anthropic) for restructuring and drafting of the text in English, for generating the code that produces the vector figures, which the authors then reviewed and edited iteratively until reaching the final versions, and for assistance in conducting and processing the literature searches reported in Appendix A. The conceptual framework, the taxonomy, and all scientific judgments are the authors' own. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: F.B. is a Guest Editor of this Special Issue but was not involved in the editorial process of this manuscript, including peer review and the decision to publish, which were handled independently by the journal. The authors declare no other conflicts of interest.

Appendix A. Search Strings

All searches were run in Scopus using the advanced document search. "Records" is the number of documents returned by the query; "Screened" is the number whose title, abstract, and venue were examined; "Recovered/new" gives two counts separated by a slash: the number of studies already identified through the concept-driven search and snowballing that the query recovered, which measures coverage rather than yield, and the number of studies added to the review as a result of the search.

Table A1. Structured search in Scopus: queries, dates of consultation, and screening counts.

ID	Stream	Search String (Scopus)	Date	Recs.	Scr.	Rec./New
S1	Predict-then-optimize, contextual optimization, decision-focused learning	TITLE-ABS-KEY (“predict-then-optimize” OR “predict then optimize” OR “smart predict” OR “decision-focused learning” OR “decision focused learning” OR “contextual optimization” OR “prescriptive analytics” OR “integrated estimation optimization”) AND PUBYEAR > 1994	27 September 2026	1530	1530	9/7
S2	Constraint learning and learned model components	TITLE-ABS-KEY (“constraint learning” OR “optimization with constraint learning” OR “learned constraints” OR “data-driven uncertainty set” OR “learning uncertainty sets”) AND PUBYEAR > 1994	30 September 2026	601	601	3/2
S3	Learning inside the solver	TITLE-ABS-KEY (“learning to branch” OR “learn to branch” OR “branching policy” OR “learning to cut” OR “cut selection” OR “primal heuristic*”) AND TITLE-ABS-KEY (“machine learning” OR “neural network*” OR “reinforcement learning” OR “imitation learning”) AND PUBYEAR > 1994	30 September 2026	127	127	4/4
S4	Algorithm selection, configuration, and design	TITLE-ABS-KEY (“algorithm selection” OR “algorithm configuration” OR “hyper-heuristic*” OR “automated algorithm design”) AND TITLE-ABS-KEY (“machine learning” OR “learning”) AND PUBYEAR > 1994	30 September 2026	2071	2071	3/2
S5	Learning in metaheuristics and guided search	TITLE-ABS-KEY ((“machine learning” OR “neural network*” OR “reinforcement learning”) AND (“metaheuristic*” OR “large neighborhood search” OR “local search” OR “dynamic programming”) AND (“combinatorial optimization” OR “routing problem*” OR “scheduling”)) AND PUBYEAR > 2009	30 September 2026	2123	2123	5/2
S6	Learned solvers	TITLE-ABS-KEY (“neural combinatorial optimization” OR “learning to optimize” OR “amortized optimization” OR “optimization proxies” OR “learned heuristic*”) AND PUBYEAR > 2014	30 September 2026	2111	595 *	8/3

Table A1. *Cont.*

ID	Stream	Search String (Scopus)	Date	Recs.	Scr.	Rec./New
S7	Algorithms with predictions	TITLE-ABS-KEY (“algorithms with predictions” OR “learning-augmented algorithm*” OR “machine learned advice”) AND PUBYEAR > 2014	2 October 2026	301	301	2/3
S8	LLMs for optimization modeling	TITLE-ABS-KEY ((“large language model*” OR “LLM” OR “GPT”) AND (“optimization model*” OR “mathematical model*” OR “mixed-integer” OR “operations research” OR “auto-formulation” OR “autoformulation”)) AND PUBYEAR > 2020	2 October 2026	706	706	5/6
S9	Explainable and interactive optimization	TITLE-ABS-KEY ((“explainable optimization” OR “explainable decision” OR “counterfactual explanation*” OR “preference elicitation” OR “preference learning”) AND (“optimization” OR “operations research” OR “mathematical programming” OR “decision support”)) AND PUBYEAR > 2004	2 October 2026	1434 **	1434	3/1

* For S6, the 300 most cited and the 300 most recent records were screened (595 unique after removing overlaps), because ranking by citations alone under-represents work published in 2025 and 2026. ** An initial S9 query that also included the unqualified terms “explainable”, “interactive”, and “machine learning” returned more than 16,000 records and was refined to the string above.

Appendix B. Notation

Table A2. Notation used in the formal framework (Section 4) and in Sections 6 and 7.

Symbol	Meaning	First Used
$z, \hat{z}$	Decision; decision produced by an AI–OR mechanism	Equation (1)
$f(z;\theta), f(z;\xi)$	Objective function; written with the instance ξ as argument when convenient	Equation (1)
$Z(\theta), Z$	Feasible set; Z when it does not depend on θ	Equation (1)
$g(z;\theta)$	Constraint functions, $Z(\theta) = \{z : g(z;\theta) \leq 0\}$	Equation (10)
$\theta, \hat{\theta}$	True uncertain parameters; predicted parameters, written as a function of the context where that dependence is made explicit	Equation (1)
$z^*(\theta)$	Optimal solution of $P(\theta)$	Equation (1)
x	Contextual information available before the decision	Section 4.1
$\mathcal{S} = \{(x_i, \theta_i)\}_{i=1}^N$	Historical dataset of N observations	Section 4.1
$\xi = (f, Z, \theta)$, Ξ, $\mathcal{D}_\Xi$	Instance; class of instances; distribution of instances; in Equation (19), the instance is assembled from the generated model and the predicted parameters	Section 4.1
A, A_π	Algorithm; algorithm run with decision rule π	Section 4.1; Equation (11)
d	Natural-language or informal problem description	Section 4.1

Table A2. *Cont.*

Symbol	Meaning	First Used
$j, r, t; z_{jrt}; \theta_j; C_{rt}$	Example 1: case, theater, day; assignment variable; case duration; session length	Equation (2)
$m_w, w, \hat{w}$	Predictor; its parameters and their fitted value (M1, M2)	Equation (3)
ℓ	Statistical training loss (M1)	Equation (3)
$\omega_i(x)$	Nonnegative weights over historical observations, summing to one	Equation (4)
$\mathcal{L}$	Decision loss (M2)	Equation (5)
$\mathrm{Regret}\left(\hat{\theta}, \theta\right)$	Regret of the decision induced by $\hat{\theta}$ under θ	Equation (6)
$\varepsilon, \zeta, z_\varepsilon^*$	Perturbation scale; random perturbation; perturbed optimizer	Equation (8)
$c, \hat{c}; l_{\mathrm{SPO+}}$	True and predicted cost vectors of a linear objective; SPO+ loss	Equation (9)
$[\cdot]_+; \lambda$	Positive part; penalty weight on constraint violation	Equation (10)
s_t, a_t	State and action at step t of an algorithm or policy	Section 4.3
$\pi, \Pi, \pi_0, \pi_\phi$	Decision rule; admissible rules; default non-learned rule; learned rule	Definition 1; Equation (11)
$C; \hat{C}_\phi$	Performance measure of an algorithm; learned performance model	Equations (11) and (12)
$\kappa, \mathcal{K}$	Algorithm configuration (or program code); configuration space	Equation (12)
ALG, OPT	Cost of a learning-augmented algorithm; optimal cost	Equation (13)
η	Error level of a learned component or prediction	Equations (13) and (21)
α, β, γ	Problem-dependent constants of consistency and robustness bounds	Equation (13)
$G_\phi; p_\phi(z \mid \xi); \pi_\phi(a_t \mid s_t)$	Learned solution generator; its distribution over solutions; constructive policy	Equations (14) and (15)
$J(\phi), b(\xi)$	Expected cost of the generator; baseline	Equation (15)
$I_\phi; q_\phi(x)$	Learned improvement operator; learned decision rule	Section 4.3
$M_\varphi; \left(\hat{f}, \hat{Z}\right)$	Learned modeling map; generated model	Section 4.3
$\hat{Z}, \hat{z}; \psi; \hat{z}^*$	Feasible set of the generated model and a solution in it; map between solutions of two models; optimal solution of the generated model	Equation (16)
ρ_k, r_k, U_φ	Preference or constraint parameters at round k; decision-maker response; learned update	Equation (17)
$N^\dagger; C_{\mathrm{data}}, C_{\mathrm{train}}; \bar{c}_{\mathrm{OR}}, \bar{c}_{\mathrm{inf}}$	Break-even number of instances; one-off data and training costs; average per-instance costs	Equation (18)
$\tilde{z}$	Exact optimum of the formulated model with predicted parameters	Equation (20)
$\Delta(\eta); \hat{z}_\eta$	Degradation profile; decision produced when the learned component has error level η	Equation (21)

References

1. Mandi, J.; Kotary, J.; Berden, S.; Mulamba, M.; Bucarey, V.; Guns, T.; Fioretto, F. Decision-Focused Learning: Foundations, State of the Art, Benchmark and Future Opportunities. *J. Artif. Intell. Res.* **2024**, *80*, 1623–1701. [CrossRef]
2. Angioni, D.; Archetti, C.; Speranza, M.G. Neural Combinatorial Optimization: A Tutorial. *Comput. Oper. Res.* **2025**, *182*, 107102. [CrossRef]
3. Ramamonjison, R.; Li, H.; Yu, T.; He, S.; Rengan, V.; Banitalebi-dehkordi, A.; Zhou, Z.; Zhang, Y. Augmenting Operations Research with Auto-Formulation of Optimization Models from Problem Descriptions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*; Association for Computational Linguistics: Abu Dhabi, United Arab Emirates, 2022; pp. 29–62.
4. Ahmaditeshnizi, A.; Gao, W.; Udell, M. OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning*; Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., Berkenkamp, F., Eds.; PMLR: London, UK, 2024; Volume 235, pp. 577–596.
5. Fan, Z.; Ghaddar, B.; Wang, X.; Xing, L.; Zhang, Y.; Zhou, Z. Artificial Intelligence for Optimization: Unleashing the Potential of Parameter Generation, Model Formulation, and Solution Methods. *Eur. J. Oper. Res.* **2026**, *332*, 1–30. [CrossRef]
6. Bengio, Y.; Lodi, A.; Prouvost, A. Machine Learning for Combinatorial Optimization: A Methodological Tour d'horizon. *Eur. J. Oper. Res.* **2021**, *290*, 405–421. [CrossRef]
7. Sadana, U.; Chenreddy, A.; Delage, E.; Forel, A.; Frejinger, E.; Vidal, T. A Survey of Contextual Optimization Methods for Decision-Making under Uncertainty. *Eur. J. Oper. Res.* **2025**, *320*, 271–289. [CrossRef]
8. Fajemisin, A.O.; Maragno, D.; den Hertog, D. Optimization with Constraint Learning: A Framework and Survey. *Eur. J. Oper. Res.* **2024**, *314*, 1–14. [CrossRef]
9. Talbi, E.-G. Machine Learning into Metaheuristics: A Survey and Taxonomy. *ACM Comput. Surv.* **2021**, *54*, 1–32. [CrossRef]
10. Karimi-Mamaghan, M.; Mohammadi, M.; Meyer, P.; Karimi-Mamaghan, A.M.; Talbi, E.-G. Machine Learning at the Service of Meta-Heuristics for Solving Combinatorial Optimization Problems: A State-of-the-Art. *Eur. J. Oper. Res.* **2022**, *296*, 393–422. [CrossRef]
11. Mazyavkina, N.; Sviridov, S.; Ivanov, S.; Burnaev, E. Reinforcement Learning for Combinatorial Optimization: A Survey. *Comput. Oper. Res.* **2021**, *134*, 105400. [CrossRef]
12. Cappart, Q.; Chételat, D.; Khalil, E.B.; Lodi, A.; Morris, C.; Veličković, P. Combinatorial Optimization and Reasoning with Graph Neural Networks. *J. Mach. Learn. Res.* **2023**, *24*, 1–61.
13. Kotary, J.; Fioretto, F.; Van Hentenryck, P.; Wilder, B. End-to-End Constrained Optimization Learning: A Survey. In *Proceedings of the International Joint Conference on Artificial Intelligence*; International Joint Conferences on Artificial Intelligence Organization: Marina del Rey, CA, USA, 2021. [CrossRef] [PubMed]
14. Mitzenmacher, M.; Vassilvitskii, S. Algorithms with Predictions. In *Beyond the Worst-Case Analysis of Algorithms*; Roughgarden, T., Ed.; Cambridge University Press: Cambridge, UK, 2021; pp. 646–662.
15. Xiao, Z.; Xie, J.; Xu, L.; Guan, S.; Zhu, J.; Han, X.; Fu, X.; Yu, W.; Wu, H.; Shi, W.; et al. A Survey of Optimization Modeling Meets LLMs: Progress and Future Directions. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*; International Joint Conferences on Artificial Intelligence Organization: Montreal, Canada, 2025; pp. 10742–10750.
16. Nickerson, R.C.; Varshney, U.; Muntermann, J. A Method for Taxonomy Development and Its Application in Information Systems. *Eur. J. Inf. Syst.* **2013**, *22*, 336–359. [CrossRef]
17. Wohlin, C. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*; Association for Computing Machinery: New York, NY, USA, 2014.
18. Bertsimas, D.; Kallus, N. From Predictive to Prescriptive Analytics. *Manag. Sci.* **2020**, *66*, 1025–1044. [CrossRef]
19. Wilder, B.; Dilkina, B.; Tambe, M. Melding the Data-Decisions Pipeline: Decision-Focused Learning for Combinatorial Optimization. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*; AAAI Press: Washington, DC, USA, 2019.
20. Berthet, Q.; Blondel, M.; Teboul, O.; Cuturi, M.; Vert, J.-P.; Bach, F. Learning with Differentiable Pertubed Optimizers. In *Proceedings of the Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 9508–9519.
21. Elmachtoub, A.N.; Grigas, P. Smart "Predict, Then Optimize". *Manag. Sci.* **2022**, *68*, 9–26. [CrossRef]
22. Zhai, H.; Lawless, C.; Vitercik, E.; Leqi, L. EquivaMap: Leveraging LLMs for Automatic Equivalence Checking of Optimization Formulations. In *Proceedings of the 42nd International Conference on Machine Learning*; JMLR: Norfolk, MA, USA, 2025.
23. Robbins, H.; Lawless, C.; Udell, M.; Vitercik, E. FLARE: Verifying MILP Reformulations with LLM-Based Theorem Proving. *arXiv* **2026**, arXiv:2608.25220.

24. Gasse, M.; Chetelat, D.; Ferroni, N.; Charlin, L.; Lodi, A. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In *Proceedings of the Advances in Neural Information Processing Systems*; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32.
25. Nair, V.; Bartunov, S.; Gimeno, F.; von Glehn, I.; Lichocki, P.; Lobov, I.; O'Donoghue, B.; Sonnerat, N.; Tjandraatmadja, C.; Wang, P.; et al. Solving Mixed Integer Programs Using Neural Networks. *arXiv* **2020**, arXiv:2012.13349.
26. Hottung, A.; Tierney, K. Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem. In *Frontiers in Artificial Intelligence and Applications*; IOS Press: Amsterdam, The Netherlands, 2020.
27. Xin, L.; Song, W.; Cao, Z.; Zhang, J. NeuroLKH: Combining Deep Learning Model with Lin-Kernighan-Helsgaun Heuristic for Solving the Traveling Salesman Problem. In *Proceedings of the Advances in Neural Information Processing Systems*; Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P.S., Vaughan, J.W., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2021; Volume 34, pp. 7472–7483.
28. Kool, W.; Van Hoof, H.; Gromicho, J.; Welling, M. Deep Policy Dynamic Programming for Vehicle Routing Problems. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*; Lecture Notes in Computer Science; Schaus, P., Ed.; Springer International Publishing: Cham, Switzerland, 2022; Volume 13292, pp. 190–213.
29. Chen, X.; Tian, Y. Learning to Perform Local Rewriting for Combinatorial Optimization. In *Proceedings of the Advances in Neural Information Processing Systems*; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32.
30. Fu, Z.-H.; Qiu, K.-B.; Zha, H. Generalize a Small Pre-Trained Model to Arbitrarily Large TSP Instances. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 7474–7482. [CrossRef]
31. Monga, V.; Li, Y.; Eldar, Y.C. Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing. *IEEE Signal Process. Mag.* **2021**, *38*, 18–44. [CrossRef]
32. Chen, T.; Chen, X.; Chen, W.; Heaton, H.; Liu, J.; Wang, Z.; Yin, W. Learning to Optimize: A Primer and A Benchmark. *J. Mach. Learn. Res.* **2022**, *23*, 1–59. [CrossRef]
33. Elmachtoub, A.N.; Liang, J.C.N.; Mcnellis, R. Decision Trees for Decision-Making under the Predict-Then-Optimize Framework. In *Proceedings of the 37th International Conference on Machine Learning*; III, H.D., Singh, A., Eds.; PMLR: London, UK, 2020; Volume 119, pp. 2858–2867.
34. Ban, G.-Y.; Rudin, C. The Big Data Newsvendor: Practical Insights from Machine Learning. *Oper. Res.* **2019**, *67*, 90–108. [CrossRef]
35. Lykouris, T.; Vassilvtiskii, S. Competitive Caching with Machine Learned Advice. In *Proceedings of the 35th International Conference on Machine Learning*; Dy, J., Krause, A., Eds.; PMLR: London, UK, 2018; Volume 80, pp. 3296–3305.
36. Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Balog, M.; Kumar, M.P.; Dupont, E.; Ruiz, F.J.R.; Ellenberg, J.S.; Wang, P.; Fawzi, O.; et al. Mathematical Discoveries from Program Search with Large Language Models. *Nature* **2024**, *625*, 468–475. [CrossRef] [PubMed]
37. Liu, F.; Xialiang, T.; Yuan, M.; Lin, X.; Luo, F.; Wang, Z.; Lu, Z.; Zhang, Q. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model. In *Proceedings of the 41st International Conference on Machine Learning*; Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., Berkenkamp, F., Eds.; PMLR: London, UK, 2024; Volume 235, pp. 32201–32223.
38. Yang, C.; Wang, X.; Lu, Y.; Liu, H.; Le, Q.V.; Zhou, D.; Chen, X. Large Language Models as Optimizers. In Proceedings of the The Twelfth International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024.
39. Lawless, C.; Schoeffer, J.; Le, L.; Rowan, K.; Sen, S.; St. Hill, C.; Suh, J.; Sarrafzadeh, B. "I Want It That Way": Enabling Interactive Decision Support Using Large Language Models and Constraint Programming. *ACM Trans. Interact. Intell. Syst.* **2024**, *14*, 1–33. [CrossRef]
40. Elmachtoub, A.N.; Lam, H.; Zhang, H.; Zhao, Y. Estimate-Then-Optimize versus Integrated-Estimation-Optimization versus Sample Average Approximation: A Stochastic Dominance Perspective. *arXiv* **2023**, arXiv:2304.06833.
41. Johnstone, C.; Cox, B. Conformal Uncertainty Sets for Robust Optimization. In *Proceedings of the Tenth Symposium on Conformal and Probabilistic Prediction and Applications*; Carlsson, L., Luo, Z., Cherubin, G., An Nguyen, K., Eds.; PMLR: London, UK, 2021; Volume 152, pp. 72–90.
42. Goerigk, M.; Kurtz, J. Data-Driven Robust Optimization Using Deep Neural Networks. *Comput. Oper. Res.* **2023**, *151*, 106087. [CrossRef]
43. Maragno, D.; Wiberg, H.; Bertsimas, D.; Birbil, Ş.İ.; den Hertog, D.; Fajemisin, A.O. Mixed-Integer Optimization with Constraint Learning. *Oper. Res.* **2025**, *73*, 1011–1028. [CrossRef]
44. Bessiere, C.; Koriche, F.; Lazaar, N.; O'Sullivan, B. Constraint Acquisition. *Artif. Intell.* **2017**, *244*, 315–342. [CrossRef]
45. De Raedt, L.; Passerini, A.; Teso, S. Learning Constraints from Examples. *Proc. AAAI Conf. Artif. Intell.* **2018**, *32*, 7965–7970. [CrossRef]
46. Bengio, Y. Using a Financial Training Criterion Rather than a Prediction Criterion. *Int. J. Neural Syst.* **1997**, *8*, 433–443. [CrossRef] [PubMed]

47. Kao, Y.; Roy, B.; Yan, X. Directed Regression. In *Proceedings of the Advances in Neural Information Processing Systems*; Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., Culotta, A., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2009; Volume 22.
48. Amos, B.; Kolter, J.Z. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*; Precup, D., Teh, Y.W., Eds.; PMLR: London, UK, 2017; Volume 70, pp. 136–145.
49. Donti, P.; Amos, B.; Kolter, J.Z. Task-Based End-to-End Model Learning in Stochastic Optimization. In *Proceedings of the Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
50. Wilder, B.; Ewing, E.; Dilkina, B.; Tambe, M. End to End Learning and Optimization on Graphs. In *Proceedings of the Advances in Neural Information Processing Systems*; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32.
51. Vlastelica, M.; Paulus, A.; Musil, V.; Martius, G.; Rolínek, M. Differentiation of Blackbox Combinatorial Solvers. *arXiv* **2020**, arXiv:1912.02175.
52. Mandi, J.; Demirović, E.; Stuckey, P.J.; Guns, T. Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 1603–1610. [CrossRef]
53. Mandi, J.; Bucarey, V.; Tchomba, M.M.K.; Guns, T. Decision-Focused Learning: Through the Lens of Learning to Rank. In *Proceedings of the 39th International Conference on Machine Learning*; Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S., Eds.; PMLR: London, UK, 2022; Volume 162, pp. 14935–14947.
54. Shah, S.; Wang, K.; Wilder, B.; Perrault, A.; Tambe, M. Decision-Focused Learning without Decision-Making: Learning Locally Optimized Decision Losses. In *Proceedings of the Advances in Neural Information Processing Systems*; Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2022; Volume 35, pp. 1320–1332.
55. Paulus, A.; Rolinek, M.; Musil, V.; Amos, B.; Martius, G. CombOptNet: Fit the Right NP-Hard Problem by Learning Integer Programming Constraints. In *Proceedings of the 38th International Conference on Machine Learning*; Meila, M., Zhang, T., Eds.; PMLR: London, UK, 2021; Volume 139, pp. 8443–8453.
56. Hu, X.; Lee, J.C.H.; Lee, J.H.M. Predict+Optimize for Packing and Covering LPs with Unknown Parameters in Constraints. *Proc. AAAI Conf. Artif. Intell.* **2023**, *37*, 3987–3995. [CrossRef]
57. Mandi, J.; Defresne, M.; Berden, S.; Guns, T. *Feasibility-Aware Decision-Focused Learning for Predicting Parameters in the Constraints*; Neural Information Processing Systems Foundation: La Jolla, CA, USA, 2025.
58. Vanderschueren, T.; Verdonck, T.; Baesens, B.; Verbeke, W. Predict-Then-Optimize or Predict-and-Optimize? An Empirical Evaluation of Cost-Sensitive Learning Strategies. *Inf. Sci.* **2022**, *594*, 400–415. [CrossRef]
59. Mulamba, M.; Mandi, J.; Diligenti, M.; Lombardi, M.; Bucarey, V.; Guns, T. Contrastive Losses and Solution Caching for Predict-and-Optimize. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*; International Joint Conferences on Artificial Intelligence Organization: Montreal, QC, Canada, 2021; pp. 2833–2840.
60. El Balghiti, O.; Elmachtoub, A.N.; Grigas, P.; Tewari, A. Generalization Bounds in the Predict-Then-Optimize Framework. *Math. Oper. Res.* **2023**, *48*, 2043–2065. [CrossRef]
61. Tang, B.; Khalil, E.B. PyEPO: A PyTorch-Based End-to-End Predict-Then-Optimize Library for Linear and Integer Programming. *Math. Program. Comput.* **2024**, *16*, 297–335. [CrossRef]
62. Khalil, E.; Le Bodic, P.; Song, L.; Nemhauser, G.; Dilkina, B. Learning to Branch in Mixed Integer Programming. *Proc. AAAI Conf. Artif. Intell.* **2016**, *30*, 1. [CrossRef]
63. Alvarez, A.M.; Louveaux, Q.; Wehenkel, L. A Machine Learning-Based Approximation of Strong Branching. *Inf. J. Comput.* **2017**, *29*, 185–195. [CrossRef]
64. Tang, Y.; Agrawal, S.; Faenza, Y. Reinforcement Learning for Integer Programming: Learning to Cut. In *Proceedings of the 37th International Conference on Machine Learning*; III, H.D., Singh, A., Eds.; PMLR: London, UK, 2020; Volume 119, pp. 9367–9376.
65. Paulus, M.B.; Zarpellon, G.; Krause, A.; Charlin, L.; Maddison, C. Learning to Cut by Looking Ahead: Cutting Plane Selection via Imitation Learning. In *Proceedings of the 39th International Conference on Machine Learning*; Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S., Eds.; PMLR: London, UK, 2022; Volume 162, pp. 17584–17600.
66. Gupta, P.; Gasse, M.; Khalil, E.; Mudigonda, P.; Lodi, A.; Bengio, Y. Hybrid Models for Learning to Branch. In *Proceedings of the Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 18087–18097.
67. Zarpellon, G.; Jo, J.; Lodi, A.; Bengio, Y. Parameterizing Branch-and-Bound Search Trees to Learn Branching Policies. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 3931–3939. [CrossRef]
68. Cantürk, F.; Varol, T.; Aydoğan, R.; Özener, O.Ö. Scalable Primal Heuristics Using Graph Neural Networks for Combinatorial Optimization. *J. Artif. Intell. Res.* **2024**, *80*, 327–376. [CrossRef]
69. Scavuzzo, L.; Aardal, K.; Lodi, A.; Yorke-Smith, N. Machine Learning Augmented Branch and Bound for Mixed Integer Linear Programming. *Math. Program.* **2026**, *217*, 123–166. [CrossRef] [PubMed]

70. Karimi-Mamaghan, M.; Mohammadi, M.; Pasdeloup, B.; Meyer, P. Learning to Select Operators in Meta-Heuristics: An Integration of Q-Learning into the Iterated Greedy Algorithm for the Permutation Flowshop Scheduling Problem. *Eur. J. Oper. Res.* **2023**, *304*, 1296–1330. [CrossRef]
71. Kallestad, J.; Hasibi, R.; Hemmati, A.; Sörensen, K. A General Deep Reinforcement Learning Hyperheuristic Framework for Solving Combinatorial Optimization Problems. *Eur. J. Oper. Res.* **2023**, *309*, 446–468. [CrossRef]
72. Kerschke, P.; Hoos, H.H.; Neumann, F.; Trautmann, H. Automated Algorithm Selection: Survey and Perspectives. *Evol. Comput.* **2019**, *27*, 3–45. [CrossRef] [PubMed]
73. Burke, E.K.; Gendreau, M.; Hyde, M.; Kendall, G.; Ochoa, G.; Özcan, E.; Qu, R. Hyper-Heuristics: A Survey of the State of the Art. *J. Oper. Res. Soc.* **2013**, *64*, 1695–1724. [CrossRef]
74. Ghaddar, B.; Gómez-Casares, I.; González-Díaz, J.; González-Rodríguez, B.; Pateiro-López, B.; Rodríguez-Ballesteros, S. Learning for Spatial Branching: An Algorithm Selection Approach. *Inf. J. Comput.* **2023**, *35*, 1024–1043. [CrossRef]
75. Himmich, I.; Er Raqabi, E.M.; El Hachemi, N.; El Hallaoui, I.; Metrane, A.; Soumis, F. MPILS: An Automatic Tuner for MILP Solvers. *Comput. Oper. Res.* **2023**, *159*, 106344. [CrossRef]
76. Ye, H.; Wang, J.; Cao, Z.; Berto, F.; Hua, C.; Kim, H.; Park, J.; Song, G. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution. In *Proceedings of the Advances in Neural Information Processing Systems*; Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., Zhang, C., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2024; Volume 37, pp. 43571–43608.
77. Çetinkaya, İ.O.; Büyüktahtakın, İ.E.; Shojaee, P.; Reddy, C.K. Discovering Heuristics with Large Language Models (LLMs) for Mixed-Integer Programs: Single-Machine Scheduling. *Comput. Oper. Res.* **2026**, *186*, 107325. [CrossRef]
78. Antoniadis, A.; Gouleakis, T.; Kleer, P.; Kolev, P. Secretary and Online Matching Problems with Machine Learned Advice. *Discret. Optim.* **2023**, *48*, 100778. [CrossRef]
79. Mitzenmacher, M. Scheduling with Predictions and the Price of Misprediction. *LIPIcs* **2020**, *151*, 14:1–14:18. [CrossRef]
80. Bamas, E.; Maggiori, A.; Svensson, O. The Primal-Dual Method for Learning Augmented Algorithms. In *Proceedings of the Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 20083–20094.
81. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer Networks. In *Proceedings of the Advances in Neural Information Processing Systems*; Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2015; Volume 28.
82. Bello, I.; Pham, H.; Le, Q.V.; Norouzi, M.; Bengio, S. Neural Combinatorial Optimization with Reinforcement Learning. *arXiv* **2016**, arXiv:1611.09940.
83. Kool, W.; van Hoof, H.; Welling, M. Attention, Learn to Solve Routing Problems! In *Proceedings of the International Conference on Learning Representations*; OpenReview: Newton Highlands, MA, USA, 2019.
84. Kwon, Y.-D.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; Min, S. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. In *Proceedings of the Advances in Neural Information Processing Systems*; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 21188–21198.
85. Bi, J.; Ma, Y.; Zhou, J.; Song, W.; Cao, Z.; Wu, Y.; Zhang, J. Learning to Handle Complex Constraints for Vehicle Routing Problems. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*; Curran Associates Inc.: Red Hook, NY, USA, 2024.
86. Wu, Y.; Song, W.; Cao, Z.; Zhang, J.; Lim, A. Learning Improvement Heuristics for Solving Routing Problems. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 5057–5069. [CrossRef] [PubMed]
87. Bi, J.; Cao, Z.; Zhou, J.; Song, W.; Wu, Y.; Zhang, J.; Ma, Y.; Wu, C. Towards Efficient Constraint Handling in Neural Solvers for Routing Problems. In Proceedings of the 14th International Conference on Learning Representations, Rio de Janeiro, Brazil, 23–27 April 2026.
88. Amos, B. Tutorial on Amortized Optimization. *Found. Trends Mach. Learn.* **2023**, *16*, 592–732. [CrossRef]
89. Donti, P.L.; Rolnick, D.; Kolter, J.Z. DC3: A Learning Method for Optimization with Hard Constraints. In Proceedings of the International Conference on Learning Representations, Virtual Conference, 3–7 May 2021.
90. Joshi, C.K.; Cappart, Q.; Rousseau, L.-M.; Laurent, T. Learning the Travelling Salesperson Problem Requires Rethinking Generalization. *Constraints* **2022**, *27*, 70–98. [CrossRef]
91. Drakulic, D.; Michel, S.; Mai, F.; Sors, A.; Andreoli, J.-M. BQ-NCO: Bisimulation Quotienting for Efficient Neural Combinatorial Optimization. In *Proceedings of the Advances in Neural Information Processing Systems*; Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2023; Volume 36, pp. 77416–77429.
92. Hou, Q.; Yang, J.; Su, Y.; Wang, X.; Deng, Y. Generalize Learned Heuristics to Solve Large-Scale Vehicle Routing Problems in Real-Time. In Proceedings of the 11th International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023.
93. Liu, S.; Zhang, Y.; Tang, K.; Yao, X. How Good Is Neural Combinatorial Optimization? A Systematic Evaluation on the Traveling Salesman Problem. *IEEE Comput. Intell. Mag.* **2023**, *18*, 14–28. [CrossRef]

94. Ramamonjison, R.; Yu, T.; Li, R.; Li, H.; Carenini, G.; Ghaddar, B.; He, S.; Mostajabdaveh, M.; Banitalebi-Dehkordi, A.; Zhou, Z.; et al. NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*; Ciccone, M., Stolovitzky, G., Albrecht, J., Eds.; PMLR: London, UK, 2022; Volume 220, pp. 189–203.
95. Xiao, Z.; Zhang, D.; Wu, Y.; Xu, L.; Wang, Y.; Han, X.; Fu, X.; Zhong, T.; Zeng, J.; Song, M.; et al. Chain-of-Experts: When LLMs Meet Complex Operations Research Problems. In *Proceedings of the International Conference on Learning Representations*; Kim, B., Yue, Y., Chaudhuri, S., Fragkiadaki, K., Khan, M., Sun, Y., Eds.; International Conference on Learning Representations (ICLR): Appleton, WI, USA, 2024; Volume 2024, pp. 48519–48537.
96. Mostajabdaveh, M.; Yu, T.T.; Ramamonjison, R.; Carenini, G.; Zhou, Z.; Zhang, Y. Optimization Modeling and Verification from Problem Specifications Using a Multi-Agent Multi-Stage LLM Framework. *INFOR Inf. Syst. Oper. Res.* **2024**, *62*, 599–617. [CrossRef]
97. Ahmed, T.; Choudhury, S. LM4OPT: Unveiling the Potential of Large Language Models in Formulating Mathematical Optimization Problems. *INFOR Inf. Syst. Oper. Res.* **2024**, *62*, 559–572. [CrossRef]
98. Huang, C.; Tang, Z.; Hu, S.; Jiang, R.; Zheng, X.; Ge, D.; Wang, B.; Wang, Z. ORLM: A Customizable Framework in Training Large Models for Automated Optimization Modeling. *Oper. Res.* **2025**, *73*, 2986–3009. [CrossRef]
99. Astorga, N.; Liu, T.; Xiao, Y.; Van Der Schaar, M. Autoformulation of Mathematical Optimization Models Using LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*; Singh, A., Fazel, M., Hsu, D., Lacoste-Julien, S., Berkenkamp, F., Maharaj, T., Wagstaff, K., Zhu, J., Eds.; PMLR: London, UK, 2025; Volume 267, pp. 1864–1886.
100. Chen, Y.; Xia, J.; Shao, S.; Ge, D.; Ye, Y. Solver-Informed RL: Grounding Large Language Models for Authentic Optimization Modeling. In *Proceedings of the Advances in Neural Information Processing Systems 38*; Neural Information Processing Systems Foundation, Inc. (NeurIPS): San Diego, CA, USA; Mexico City, Mexico, 2025; pp. 117446–117488.
101. Yang, Z.; Wang, Y.; Huang, Y.; Guo, Z.; Shi, S.; Han, X.; Feng, L.; Song, L.; Liang, X.; Tang, J. OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling. In *Proceedings of the International Conference on Learning Representations*; Yue, Y., Garg, A., Peng, N., Sha, F., Yu, R., Eds.; International Conference on Learning Representations (ICLR): Singapore, 2025; Volume 2025, pp. 24726–24759.
102. Chen, H.; Constante-Flores, G.E.; Mantri, K.S.I.; Kompalli, S.M.; Ahluwalia, A.S.; Li, C. OptiChat: Bridging Optimization Models and Practitioners with Large Language Models. *Inf. J. Data Sci.* **2026**, *5*, 199–220. [CrossRef]
103. Zhang, Y.; Kang, Q.; Yu, W.Y.; Gong, H.; Fu, X.; Han, X.; Zhong, T.; Ma, C. Decision Information Meets Large Language Models: The Future of Explainable Operations Research. In *Proceedings of the International Conference on Learning Representations*; Yue, Y., Garg, A., Peng, N., Sha, F., Yu, R., Eds.; International Conference on Learning Representations (ICLR): Singapore, 2025; Volume 2025, pp. 65698–65722.
104. Battiti, R.; Passerini, A. Brain–Computer Evolutionary Multiobjective Optimization: A Genetic Algorithm Adapting to the Decision Maker. *IEEE Trans. Evol. Comput.* **2010**, *14*, 671–687. [CrossRef]
105. Branke, J.; Greco, S.; Slowinski, R.; Zielniewicz, P. Learning Value Functions in Interactive Evolutionary Multiobjective Optimization. *IEEE Trans. Evol. Comput.* **2015**, *19*, 88–102. [CrossRef]
106. Branke, J.; Deb, K.; Miettinen, K.; Slowinski, R. *Multiobjective Optimization: Interactive and Evolutionary Approaches*; Springer: Berlin/Heidelberg, Germany, 2008.
107. Korikov, A.; Beck, J.C. Counterfactual Explanations via Inverse Constraint Programming. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*; Michel, L.D., Ed.; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2021; Volume 210, pp. 35:1–35:16.
108. Forel, A.; Parmentier, A.; Vidal, T. Explainable Data-Driven Optimization: From Context to Decision and Back Again. In *Proceedings of the 40th International Conference on Machine Learning*; Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J., Eds.; PMLR: London, UK, 2023; Volume 202, pp. 10170–10187.
109. Wu, L.; Kong, L.; He, F.; Wang, B.; Su, Y.; Wang, X.; Jiang, X. Backdooring Rationalization Plus. *Neural Netw.* **2027**, *205*, 109556. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.