Les Cahiers du GERAD ISSN: 0711–2440

Large language model-driven automated design of truck dispatching rules for open-pit mines under uncertainty

J. Luo, R. Dimitrakopoulos

G–2026–72

October 2026

La collection *Les Cahiers du GERAD* est constituée des travaux de recherche menés par nos membres. La plupart de ces documents de travail a été soumis à des revues avec comité de révision. Lorsqu'un document est accepté et publié, le pdf original est retiré si c'est nécessaire et un lien vers l'article publié est ajouté.

Citation suggérée : J. Luo, R. Dimitrakopoulos (Octobre 2026). Large language model-driven automated design of truck dispatching rules for open-pit mines under uncertainty, Rapport technique, Les Cahiers du GERAD G– 2026–72, GERAD, HEC Montréal, Canada.

Avant de citer ce rapport technique, veuillez visiter notre site Web (https://www.gerad.ca/fr/papers/G-2026-72) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

The series *Les Cahiers du GERAD* consists of working papers carried out by our members. Most of these pre-prints have been submitted to peer-reviewed journals. When accepted and published, if necessary, the original pdf is removed and a link to the published article is added.

Suggested citation: J. Luo, R. Dimitrakopoulos (October 2026). Large language model-driven automated design of truck dispatching rules for open-pit mines under uncertainty, Technical report, Les Cahiers du GERAD G–2026–72, GERAD, HEC Montréal, Canada.

Before citing this technical report, please visit our website (https://www.gerad.ca/en/papers/G-2026-72) to update your reference data, if it has been published in a scientific journal.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2026
– Bibliothèque et Archives Canada, 2026

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2026
– Library and Archives Canada, 2026

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Large language model-driven automated design of truck dispatching rules for open-pit mines under uncertainty

Jingyu Luo

Roussos Dimitrakopoulos

Department of Mining and Materials Engineering & COSMO Stochastic Mine Planning Laboratory, McGill University, Montreal (Qc), Canada H3A 0E8

GERAD, Montréal (Qc), Canada, H3T 1J4

jingyu.luo@mcgill.ca
roussos.dimitrakopoulos@mcgill.ca

October 2026
Les Cahiers du GERAD
G–2026–72

Copyright © 2026 Luo, Dimitrakopoulos

Les textes publiés dans la série des rapports de recherche *Les Cahiers du GERAD* n'engagent que la responsabilité de leurs auteurs. Les auteurs conservent leur droit d'auteur et leurs droits moraux sur leurs publications et les utilisateurs s'engagent à reconnaître et respecter les exigences légales associées à ces droits. Ainsi, les utilisateurs:

- Peuvent télécharger et imprimer une copie de toute publication du portail public aux fins d'étude ou de recherche privée;
- Ne peuvent pas distribuer le matériel ou l'utiliser pour une activité à but lucratif ou pour un gain commercial;
- Peuvent distribuer gratuitement l'URL identifiant la publication.

Si vous pensez que ce document enfreint le droit d'auteur, contactez-nous en fournissant des détails. Nous supprimerons immédiatement l'accès au travail et enquêterons sur votre demande.

The authors are exclusively responsible for the content of their research papers published in the series *Les Cahiers du GERAD*. Copyright and moral rights for the publications are retained by the authors and the users must commit themselves to recognize and abide the legal requirements associated with these rights. Thus, users:

- May download and print one copy of any publication from the public portal for the purpose of private study or research;
- May not further distribute the material or use it for any profit-making activity or commercial gain;
- May freely distribute the URL identifying the publication.

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Abstract : Real-time truck dispatching in open-pit mines requires repeated truck assignments while accounting for production requirements, equipment congestion, and uncertain ore grades. Repeated optimization or search can be computationally demanding when decisions must be updated frequently, whereas dispatching rules provide a computationally inexpensive alternative. However, designing a high-performing rule may require substantial domain knowledge and repeated evaluation. Recent advances in the code-generation capabilities of large language models (LLMs) enable executable rules to be iteratively generated and revised using performance feedback.

This paper presents an LLM-driven evolutionary framework for automated truck-dispatching-rule design. During offline search, the LLM generates and revises candidate rules, while discrete-event simulation under geological and operational uncertainty determines fitness and guides subsequent revisions. The best rule found is applied directly to unseen instances without further search or LLM calls.

Experiments compare the LLM-generated rules with those generated by genetic programming (GP) and fourteen manually designed rules on standard and large-instance test pools. On standard instances, the LLM-generated rules achieve numerically better average performance and lower variation across independent runs than the GP-generated rules. The LLM search requires substantially fewer recorded evaluations and less computation time. On substantially larger unseen instances, the standard-trained LLM rules significantly outperform the standard-trained GP rules and all manually designed baselines. Rules trained at either scale remain effective when applied directly at the other, demonstrating strong generalization across instance scales. Additional analyses examine configuration effects and show that the large-instance advantage over GP persists across the tested objective weights.

Keywords : Truck dispatching ; open-pit mining ; large language models ; automated heuristic design ; dispatching rules ; geological uncertainty

Acknowledgements: This work was supported in part by IVADO Research Regroupement 9; the Natural Sciences and Engineering Research Council of Canada (NSERC) [grant number 239019]; the industry consortium supporting McGill University's COSMO Stochastic Mine Planning Laboratory; and the Canada Research Chairs Program.

1 Introduction

Real-time truck dispatching coordinates material transport from shovels to processing facilities and waste dumps in open-pit mines. Whenever a truck becomes available, a muck pile is selected as the material source for its next trip. These repeated assignments affect equipment waiting times and the quantity and grade of ore delivered to the mill (Alarie and Gamache, 2002). Decisions must account for uncertain ore grades, variable operating times, and equipment availability. Effective dispatching therefore requires a balance between productive equipment use and the quantity and grade of material delivered to the mill.

Truck dispatching and fleet allocation have been addressed using mathematical optimization, metaheuristic search, and rule-based dispatching (Alarie and Gamache, 2002; Moradi Afrapoli et al., 2019; Both and Dimitrakopoulos, 2020; Smith et al., 2021). Optimization models coordinate assignments with production requirements, while metaheuristics offer flexible search procedures for complex planning and allocation problems. Repeated optimization or search can nevertheless be computationally demanding when dispatching decisions must be updated frequently under uncertainty. Dispatching rules make assignments directly using observable operating information. Their low computational cost and explicit criteria make them suitable for real-time dispatching, either independently or within multi-stage systems that combine production planning or flow allocation with rule-based dispatching (Munirathinam and Yingling, 1994; Alarie and Gamache, 2002).

The performance of manually designed rules depends on their choice of inputs and how competing criteria are combined. Rules based on queue length or trip duration alone may not adequately control mill-feed quantity and grade. Designing composite rules requires domain knowledge and repeated evaluation, and changes in fleet composition, material characteristics, or production requirements may call for further adjustment. These demands motivate automated rule design. Reinforcement learning has been used to learn state-dependent truck assignments and related production decisions in mining (de Carvalho and Dimitrakopoulos, 2021; Noriega et al., 2025). Related work also incorporates incoming ore-quality information into updated orebody models to support equipment allocation and production decisions (de Carvalho and Dimitrakopoulos, 2023). A trained policy can make decisions without online optimization, although neural-network policies are difficult to inspect directly. Genetic programming (GP) offers an explicit rule-design approach by evolving priority functions over problem features (Branke et al., 2015; Nguyen et al., 2017). These functions can be inspected and reused, but the search may require many candidate evaluations, especially when each evaluation involves simulation across multiple instances and uncertain conditions.

Large language models (LLMs) provide another mechanism for generating and revising executable heuristics from problem descriptions and performance feedback (Liu et al., 2024; Ye et al., 2024). Recent studies have extended LLM-driven evolutionary search to dynamic manufacturing scheduling (Huang et al., 2025; Qiu et al., 2026). This form of search can generate explicit executable rules that, once selected, can be applied without further LLM calls. However, the use of LLM-driven rule design for real-time truck dispatching in open-pit mines remains largely unexplored. Real-time truck assignment must account for production progress, ore-grade information, and equipment congestion. Candidate piles are evaluated using pre-shift grade estimates, and after each ore delivery, the tonnage-weighted average grade of mill feed is updated using the realized grade of that load. Subsequent assignments therefore respond to the updated mill-feed grade, production progress, and equipment congestion. The study therefore asks two questions: whether LLM-driven search can generate effective dispatching rules under geological and operational uncertainty with manageable offline computational effort, and whether the resulting rules remain effective when applied unchanged to unseen instances, including instances substantially larger than those used for training.

To address these questions, this paper makes three contributions. First, it presents an LLM-driven evolutionary framework for offline truck-dispatching-rule design under geological and operational uncertainty. Second, it systematically evaluates rule quality, computational effort, variation across

independent search runs, and generalization to unseen instances of different scales. Third, it examines the robustness and operational behavior of the generated rules through objective decomposition, sensitivity analysis, and rule-level interpretation.

The remainder of the paper is organized as follows. Section 2 reviews truck dispatching, automated dispatching-rule design, and LLM-driven heuristic design. Section 3 formulates the dispatching problem and its objective. Section 4 describes the LLM-driven search, and Section 5 specifies the experimental design. Section 6 presents the results, and Section 7 concludes the paper and discusses future work.

2 Related work and motivation

This section reviews truck dispatching under uncertainty and methods for automatically designing dispatching rules. Section 2.1 reviews truck-dispatching approaches in open-pit mines and their treatment of geological and operational uncertainty. Section 2.2 reviews automated dispatching-rule design in mining and related scheduling problems. Section 2.3 reviews LLM-driven heuristic design and establishes the research motivation and scope of the present study.

2.1 Truck dispatching in open-pit mines

Truck-dispatching approaches can be classified as single-stage or multi-stage according to how material-flow planning and real-time truck assignments are organized (Munirathinam and Yingling, 1994; Alarie and Gamache, 2002). Single-stage approaches make assignments directly from current operating information, using criteria such as shovel waiting time, truck waiting time, or cycle time, without a separate optimization stage for material-flow targets. Multi-stage systems, including the DISPATCH system described by White and Olson (1986), first establish target material flows or equipment allocations and then coordinate individual truck assignments with these targets.

Within these dispatching structures, individual assignments must balance equipment utilization with production requirements. Optimization-based dispatching methods address this coordination problem by explicitly accounting for waiting times and deviations from planned material flows. Moradi Afrapoli et al. (2019) formulate dynamic truck allocation as a multiple-objective transportation problem that considers both types of criteria. Smith et al. (2021) propose dispatching policies based on nonlinear flow-rate optimization with queueing effects and time-discretized mixed integer programming. Their policies account for queueing and blending requirements, with a heuristic solution procedure that supports using the mixed-integer formulation in real-time operations. Mirzaei-Nasirabad et al. (2023) combine allocation planning with dynamic truck allocation to reduce fleet waiting times and deviations from planned production.

The ability to meet production requirements also depends on equipment conditions and the grades of the material supplied to the mill. Equipment failures can reduce loading or hauling capacity, while uncertain grades affect the composition of delivered mill feed. To address equipment faults, Shi et al. (2026) combine production scheduling with dynamic reassignment of transportation tasks based on observed equipment status, using a hybrid multi-objective particle swarm and genetic algorithm. Beyond equipment-related uncertainty, ore grades are also subject to geological uncertainty. This uncertainty can be represented by sets of spatially correlated orebody realizations generated through geostatistical simulation (Goovaerts, 1997; Dimitrakopoulos and Yao, 2023). These realizations have been incorporated into short-term production and fleet planning. Villalba Matamoros and Dimitrakopoulos (2016) formulate a stochastic integer programming model that jointly considers production sequencing, fleet allocation, and blending restrictions under uncertainty in orebody attributes and equipment availability. Both and Dimitrakopoulos (2020) jointly optimize extraction sequences, shovel relocation, truck-fleet scheduling, and downstream material allocation under geological and equipment-performance uncertainty, using simulated annealing to solve the resulting model. These planning-level studies coordinate fleet decisions with the material selected and scheduled for extraction.

The need to update decisions as new operating information becomes available has also motivated learning-based approaches (Hazrathosseini and Moradi Afrapoli, 2024). Kumar et al. (2020) develop adaptive learning mechanisms for revising short-term production decisions, while de Carvalho and Dimitrakopoulos (2021) integrate production planning and truck dispatching through reinforcement learning under uncertainty. de Carvalho and Dimitrakopoulos (2023) combine actor-critic reinforcement learning with the updating of simulated orebody models using incoming ore-quality measurements. The updated information supports shovel allocation, truck allocation, and material-destination decisions. At the short-term production-planning level, learning-based methods have also been used to coordinate equipment allocation with material-quality decisions. de Carvalho and Dimitrakopoulos (2024) use actor-critic reinforcement learning to jointly determine shovel allocation and grade-control decisions. For real-time truck assignment, Noriega et al. (2025) train a dispatching policy using double deep Q-learning in a discrete-event simulation based on historical operating data. The simulation accounts for variability in truck and loading-equipment performance during policy training.

2.2 Automated design of dispatching rules

Automated dispatching-rule design seeks to learn decision functions that can be reused across scheduling instances and operating states. Related work includes learning priorities from existing schedules, searching directly over rule expressions, and learning dispatching policies through interaction with a scheduling environment (Branke et al., 2015; Zhang et al., 2024). These approaches differ in the information used during learning, the representation of the resulting decision function, and the computational effort required to obtain it.

Supervised learning derives dispatching decisions or priority values from reference schedules. For example, Jun and Lee (2021) learn decision-tree dispatching rules for single-machine scheduling with dynamic arrivals. In resource-constrained project scheduling, Luo et al. (2026) train regression models on activity features and priority values derived from existing schedules; the predictions then guide schedule generation for unseen projects. Such methods use scheduling knowledge contained in reference solutions but require suitable training examples, whose generation can itself be computationally demanding. Their interpretability depends on the learned representation: a compact decision tree or regression expression is more directly inspectable than a large ensemble or neural network.

Within evolutionary hyper-heuristics, GP searches directly over rule structures and assesses candidates through the schedules or simulated operations they generate. Its applications include single-machine, parallel-machine, job-shop, and flexible job-shop scheduling, covering both sequencing and machine-assignment decisions (Branke et al., 2015; Nguyen et al., 2017). Its flexible representation allows combinations of problem features and mathematical operations to be evolved for different objectives and operating conditions. For example, Ferreira et al. (2022) develop compact rules for dynamic job shops and examine their performance under different utilization levels, due-date settings, and stochastic processing times. This approach does not require labeled dispatching decisions, but repeated candidate evaluation can make the offline search computationally expensive. Controlling the size and structural complexity of the generated rule expressions remains important for keeping the final rules interpretable.

Beyond the mining applications reviewed in Section 2.1, reinforcement learning has been used both to learn dispatching decisions directly and to select among predefined dispatching rules. Zhang et al. (2020) and Park et al. (2021) combine graph neural networks with deep reinforcement learning to learn which operation to schedule next in job-shop problems. The graph neural networks encode the current scheduling state, and the learned policies select operations using these representations. Both studies examine generalization to unseen instances of different sizes. For dynamic job shops, Zeng et al. (2022) learn a policy that selects a dispatching rule according to the current operating state. These approaches learn state-dependent decisions through interaction with a scheduling environment without requiring reference schedules. However, the learned decision mapping is represented by a neural network, making its operational logic less directly accessible than a compact symbolic expression. This distinction also

applies to rule-selection methods: the available rules may be explicit, while the mechanism that selects among them is still represented by a neural network.

For practical dispatching, the representation of the learned decision function matters alongside its performance. Compact explicit rules let engineers trace how operational inputs affect priority scores and check whether the resulting logic aligns with operating requirements (Nguyen et al., 2017; Ferreira et al., 2022). In mining, this includes checking how production progress, grade feedback, and congestion influence truck assignments. Compact explicit dispatching rules can be implemented as self-contained scoring functions with a small number of arithmetic and conditional operations. These functions require little computation at each dispatch event and do not involve online optimization.

2.3 LLM-driven heuristic design and research motivation

Advances in LLM code-generation capabilities have motivated their use in automated heuristic design (Liu et al., 2024; Ye et al., 2024). Given a problem description and implementation requirements, an LLM can generate candidate heuristics as executable programs. Researchers can then evaluate their performance and use it as feedback to guide subsequent revisions. FunSearch (Romera-Paredes et al., 2024) maintains a database of programs and generates new candidates from sampled examples. EoH (Liu et al., 2024) combines program representations with natural-language descriptions and uses several prompt strategies to generate or modify heuristics. ReEvo (Ye et al., 2024) introduces reflective feedback on performance differences between candidate heuristics. LLaMEA (Van Stein and Bäck, 2024) applies LLM-driven evolution to the generation and refinement of metaheuristic algorithms.

Research on LLM-driven heuristic design has also examined generalization to unseen instances and applications to dynamic scheduling. For example, LLaMEA tests generated algorithms at problem dimensions beyond those used during design (Van Stein and Bäck, 2024). Çetinkaya et al. (2026) adapt FunSearch to discover heuristics for the single-machine total tardiness problem. Their approach refines existing heuristics through LLM-generated code and performance evaluation, and the resulting heuristics are tested on instances substantially larger than those used for training. Huang et al. (2025) combine LLM-driven rule evolution with model fine-tuning for human-robot collaborative flexible manufacturing and evaluate the resulting approach against GP, gene expression programming, and other scheduling methods. Qiu et al. (2026) propose EvoDR for dynamic flexible assembly flow shop scheduling, using separate LLM roles for code generation and scheduling analysis to evolve dispatching rules. LLM guidance has also been incorporated into the search procedures of scheduling algorithms. Chen (2026) combines LLM guidance with coupling-aware destruction in a multiobjective iterated greedy algorithm for energy-efficient blocking hybrid flow shop scheduling.

For mine dispatching, ore-grade outcomes and production requirements must be considered alongside equipment congestion. Existing applications of LLM-driven rule design to manufacturing scheduling provide a methodological basis, but evidence on the performance of LLM-generated truck-dispatching rules under geological and operational uncertainty remains limited. This motivates examining whether effective truck-dispatching rules can be generated with manageable computational effort and whether they generalize to unseen instances, including substantially larger instances, without redesign.

The present study examines these questions for real-time truck assignment within a production shift. Shovel allocations, material destinations, and production targets are specified in advance, while truck assignments respond to production progress, delivered-grade feedback, and congestion. The setting includes geological uncertainty in ore grades, stochastic cycle times, and shift-level equipment availability. Rules are designed offline and subsequently applied to unseen instances without further search or LLM calls.

3 Problem formulation

This section formulates the real-time truck-dispatching problem considered in this study. It defines the system inputs, uncertainty sources, state variables, dispatching decisions, and simulation transitions, followed by the objective function and operational conditions governing feasible assignments.

3.1 Problem description

Consider an open-pit mine operating over a production shift of duration H (hours). Material is excavated from muck piles by shovels, loaded onto trucks, and transported to material-receiving destinations. Each muck pile represents the smallest schedulable extraction unit and may correspond to one or more underlying mining blocks; its tonnage and grade are aggregate values over the constituent blocks. Shovel allocation to muck piles is given as input and fixed throughout the shift. The truck dispatching problem is formulated as a sequential decision problem under uncertainty. At each dispatch event, occurring at the start of the shift for an initially available truck or when an operational truck completes dumping and becomes idle before H, a dispatching rule selects a feasible muck pile for the truck to visit next. The truck then travels unloaded to the selected muck pile, is loaded by the assigned shovel, and hauls the material to a destination determined by the ore/waste classification of the muck pile: ore to the processing mill, waste to the dump. The study considers a milling operation served by one processing mill and one waste dump. This two-destination setting represents the truck-assignment problem for a milling configuration while retaining competition between ore and waste haulage for the truck fleet, equipment congestion, and the joint control of mill-feed tonnage and ore grade. Additional destinations, such as leach pads, may be used depending on ore grades, mineralogical characteristics, and the processing routes available at the operation (de Carvalho and Dimitrakopoulos, 2021). Accordingly, the present formulation focuses on the mill–dump configuration as a representative setting for the milling operation considered here.

The problem is subject to multiple sources of uncertainty. The ore grade of each muck pile is uncertain and is represented by a set of geological scenarios, with the realized grade varying from scenario to scenario. Operational cycle times (loading, hauling, returning, and dumping) likewise vary stochastically around their nominal values, and equipment may be unavailable at the start of a shift due to mechanical failure or maintenance. The dispatching rule must perform well across these uncertain conditions.

3.2 Sets and indexes

$\mathbb{M}$	Set of muck piles, indexed by m
$\mathbb{D}$	Set of material-receiving destinations, $\mathbb{D} = \{\text{mill}, \text{dump}\}$, indexed by d
$\mathbb{S}$	Set of shovels, indexed by s
$\mathbb{T}$	Set of trucks, indexed by t
$\mathbb{E}$	Set of dispatch events, indexed chronologically by e
$Pred(m)$	Set of predecessor muck piles that must be depleted before m is accessible
$s(m)$	Allocated shovel of muck pile m, defined only for muck piles with an assigned shovel
$\mathbb{O}$	Set of truck origin nodes for unloaded travel, $\mathbb{O} = \{\text{mill}, \text{dump}, \text{park}\}$, indexed by o
Ω	Set of geological scenarios, indexed by ω; each scenario gives a grade for every muck pile

3.3 Static input parameters

The following parameters are fixed before the shift begins and remain unchanged during the simulation.

Mine and production parameters

TC_m	Initial tonnage of muck pile m (tons)
C_m	Pre-shift ore/waste classification of muck pile m, $C_m \in \{\text{ore}, \text{waste}\}$
H	Shift horizon for initiating new dispatches (hours)
pt	Mill-feed tonnage target (tons/shift)
gt	Target average ore grade of mill feed (grade fraction)
up	Upper mill-feed tonnage bound (tons/shift)
lp	Lower mill-feed tonnage bound (tons/shift)
ug	Upper bound on the average ore grade of mill feed (grade fraction)
lg	Lower bound on the average ore grade of mill feed (grade fraction)

The mill-feed tonnage target pt specifies the quantity of ore to be delivered during the shift, while gt specifies its target tonnage-weighted average grade. The bounds lp and up refer to mill-feed tonnage, and lg and ug refer to its average ore grade. The targets specify the desired operating point, while the bounds define reference ranges for normalizing deviations.

Grade values are expressed here as mass fractions of the metal of interest. For example, a copper grade of 0.0035 corresponds to 0.35% copper by mass.

Equipment parameters

$W_{s,m}$	Binary, 1 if shovel s is allocated to muck pile m (given as input)
EC_t	Payload capacity of truck t (tons)
$\bar{LT}_{t,s}$	Nominal loading time for truck t at shovel s (hours)
$\bar{HT}_{t,m,d}$	Nominal haul (loaded) travel time of truck t from muck pile m to destination d (hours)
$\bar{RT}_{t,o,m}$	Nominal return (unloaded) travel time of truck t from origin node o to muck pile m (hours)
$\bar{DT}_{t,d}$	Nominal dumping time of truck t at destination d (hours)
CD_d	Service capacity at destination d (number of trucks that can dump simultaneously)
SA_s	Availability rate of shovel s (probability of being operational during a shift)
TA_t	Availability rate of truck t (probability of being operational during a shift)

Objective weight parameters

λ_P	Weight on the production-shortfall penalty
λ_G^-	Weight on the below-target grade penalty
λ_G^+	Weight on the above-target grade penalty
λ_C	Weight on the mill capacity-violation penalty

These weights act on the normalized penalty terms and set their importance relative to the contained-metal term in the objective function; operators can set them to express their own priorities. The values used in the main experiments are stated in Section 5.5.

Destination mapping. Each muck pile carries a pre-shift ore/waste classification $C_m \in \{\text{ore}, \text{waste}\}$, determined before the shift by grade control and held fixed across all geological scenarios. The destination for material from muck pile m follows from this classification:

$$dest(m) = \begin{cases} \text{mill}, & \text{if } C_m = \text{ore}, \\ \text{dump}, & \text{if } C_m = \text{waste}. \end{cases} \tag{1}$$

The study focuses on real-time truck assignments with material destinations specified before the shift. The evaluated setting comprises one processing mill and one waste dump, and the dispatching rule selects the source pile for each truck's next trip.

3.4 Uncertainty model

The model considers two categories of uncertainty: geological uncertainty in muck-pile grades and operational uncertainty in operation durations and equipment availability. Ore grades are represented by a finite set of scenarios, operation durations are sampled from probability distributions, and equipment availability is modeled using independent shift-level Bernoulli trials.

3.4.1 Geological uncertainty

The grade of each muck pile is subject to geological uncertainty arising from incomplete knowledge of the orebody being mined. Let $\Omega = \{\omega_1, \ldots, \omega_K\}$ denote a finite set of equally probable geological scenarios. Each scenario ω assigns a grade $G_m(\omega)$ to every muck pile m; within a scenario, these grades are spatially correlated. The scenario-generation procedure used in the computational study is specified in Section 5.2.

Material destinations are treated as inputs to real-time truck assignment, consistent with the predefined block destinations in de Carvalho and Dimitrakopoulos (2021). In the present study, the ore/waste classification C_m, its associated destination $dest(m)$, and the grade target and bounds are specified before the shift and held fixed across geological scenarios. Each scenario supplies a possible set of realized pile grades $G_m(\omega)$. A pile classified as ore therefore remains mill-bound even if its realized grade falls below the pre-shift classification threshold. Candidate rules use pre-shift estimated pile grades and the cumulative delivered-grade state to score candidate piles at each dispatch event. The delivered-grade state is updated using realized grades after mill deliveries and therefore reflects the material delivered so far. Geological uncertainty thus affects contained-metal delivery, grade deviations, and subsequent truck assignments under the given destination plan.

3.4.2 Operation-time variability

The durations of loading, hauling, returning, and dumping operations are stochastic. A log-normal distribution is used for individual operation durations because it ensures strictly positive values and accommodates right-skewed variability. Log-normal models have also been fitted to empirical truck-shovel activity times in mining dispatch simulations (Chaowasakoo et al., 2017). For each individual truck operation within a trip, the realized duration is sampled independently from this distribution, with mean equal to the corresponding nominal duration ($\bar{LT}_{t,s}$ for loading, and analogously $\bar{HT}_{t,m,d}$, $\bar{RT}_{t,o,m}$, and $\bar{DT}_{t,d}$) and coefficient of variation CV. A single CV is shared across all operation types, so that operations are subject to the same level of relative variability regardless of their nominal duration. These nominal durations are fixed input parameters, and the realized operation durations vary randomly around them.

3.4.3 Equipment availability

Equipment availability is modeled at the shift level. At the start of each simulated shift, the operational status of each piece of equipment is determined by an independent Bernoulli trial:

$$A_s \sim \text{Bernoulli}(SA_s) \quad \forall s \in \mathbb{S}, \qquad A_t \sim \text{Bernoulli}(TA_t) \quad \forall t \in \mathbb{T}. \tag{2}$$

Equipment for which $A = 0$ is excluded from the shift entirely. Muck piles whose assigned shovel is unavailable are removed from the feasible set for the duration of the shift. Availability is resolved once per shift and held fixed; within-shift breakdowns and repairs are not represented in the current model.

3.5 Dynamic state variables

The following variables describe the system state at the time τ_e of dispatch event e. These state variables evolve as operations proceed during the shift.

τ_e	Time at which dispatch event e occurs (hours)
$RTC_m(\tau_e)$	Remaining tonnage at muck pile m (tons)
$QL_s(\tau_e)$	Number of trucks currently waiting or being served at shovel s
$QL_d(\tau_e)$	Number of trucks currently waiting or being served at destination d
$PROD(\tau_e)$	Cumulative mill-feed tonnage delivered up to time τ_e (tons)
$GRAD(\tau_e)$	Tonnage-weighted average grade of material delivered to the mill up to time τ_e (grade fraction)
$loc_t(\tau_e)$	Current origin node of truck t for the next unloaded trip; initially $loc_t(0) = \text{park}$

For truck t_e dispatched at event e, the operationally feasible muck-pile set is

$$\mathbb{M}^{feas}(\tau_e, t_e) = \big\{m \in \mathbb{M} : RTC_m(\tau_e) > 0,\ s(m) \text{ is defined},\ A_{s(m)} = 1, \\ RTC_{m'}(\tau_e) = 0 \quad \forall m' \in Pred(m)\big\}. \tag{3}$$

In the base formulation, a muck pile remains feasible even when $RTC_m(\tau_e) < EC_{t_e}$, and the final positive load from that muck pile may be smaller than the truck's capacity. This allows extraction of residual tonnage.

3.6 Decision variable and dispatch rule

At each dispatch event e with $\tau_e < H$, the dispatching decision for the idle truck t_e is the selection of a feasible muck pile x_e for its next trip. In this work, the dispatch rule takes the form of a priority scoring function $h : \mathbb{R}^n \to \mathbb{R}$, which assigns a scalar score to each feasible muck pile based on observable state features. At dispatch time, only muck piles that satisfy the operational feasibility conditions in $\mathbb{M}^{feas}(\tau_e, t_e)$ and the following time-feasibility screening criterion are considered:

$$\bar{RT}_{t_e, loc_{t_e}(\tau_e), m} + \bar{LT}_{t_e, s(m)} + \bar{HT}_{t_e, m, dest(m)} + \bar{DT}_{t_e, dest(m)} \le H - \tau_e. \tag{4}$$

This screening uses nominal operation durations because realized durations are not known at dispatch time. It excludes a candidate when its nominal cycle exceeds the remaining time before H. This screening does not include queueing times. A trip that passes the screening may therefore finish after H because of queueing delays or stochastic variation in operation durations. Trips already underway are allowed to complete, and their deliveries are counted. No new dispatch decisions are initiated at or after H.

Among the screened candidate piles, the selected muck pile is

$$x_e = \arg \max_{m \in \mathbb{M}^{feas}(\tau_e, t_e)} h(\phi(\tau_e, t_e, m)), \tag{5}$$

where the maximization is restricted to candidate piles satisfying the time-feasibility condition above. If no pile satisfies both the operational and time-feasibility conditions, the truck receives no assignment and takes no further trips during the shift. Here, $\phi(\tau_e, t_e, m) \in \mathbb{R}^n$ is the feature vector constructed from observable state variables at time τ_e, the attributes of truck t_e, and the attributes of candidate muck pile m. The detailed composition of ϕ is specified in Section 4.2.

3.7 State transitions

Given the dispatch decision $x_e = m$ for truck t_e at time τ_e, the system state evolves through the following sequence of events. The objective is evaluated by simulating these transitions with a discrete-event simulation (DES). All operation durations below are realized values sampled from the distributions defined in the operational uncertainty model.

1. Truck t_e travels unloaded from its origin $o_e = loc_{t_e}(\tau_e)$ to muck pile m, with realized duration $RT_{t_e, o_e, m}$.

2. Truck t_e joins the queue at shovel $s(m)$. If the shovel is occupied, the truck waits until service begins.
3. Truck t_e undergoes loading service at shovel $s(m)$, with realized duration $LT_{t_e,s(m)}$.
4. The realized load carried by truck t_e is

$$Load_e = \min\{EC_{t_e}, RTC_m^{\text{load}-}\}, \tag{6}$$

where $RTC_m^{\text{load}-}$ denotes the remaining tonnage at muck pile m immediately before loading begins. The truck carries a full payload when the pile holds at least EC_{t_e}; otherwise, it carries the remaining tonnage. If other trucks have depleted the pile before loading begins, $Load_e = 0$, and the truck completes the same travel and destination-service steps without delivering material.

5. After loading, the remaining tonnage at muck pile m is updated as

$$RTC_m \leftarrow RTC_m - Load_e. \tag{7}$$

6. Truck t_e travels to $dest(m)$, with realized duration $HT_{t_e,m,dest(m)}$.
7. Truck t_e joins the queue at destination $dest(m)$. If all $CD_{dest(m)}$ service slots are occupied, the truck waits in a first-come, first-served queue until a slot becomes available.
8. Truck t_e completes destination service, with realized duration $DT_{t_e,dest(m)}$.
9. If $dest(m) = \text{mill}$ and $Load_e > 0$, the delivered material is recorded using the realized scenario grade $G_m(\omega)$ of muck pile m under the current scenario ω. Let $PROD^{old}$ and $GRAD^{old}$ denote the cumulative mill tonnage and tonnage-weighted average grade before this delivery. The new cumulative mill tonnage is

$$PROD^{new} = PROD^{old} + Load_e, \tag{8}$$

and the tonnage-weighted average grade is updated as

$$GRAD^{new} = \begin{cases} G_m(\omega), & \text{if } PROD^{old} = 0, \\ \dfrac{GRAD^{old} \cdot PROD^{old} + Load_e \cdot G_m(\omega)}{PROD^{new}}, & \text{if } PROD^{old} > 0. \end{cases} \tag{9}$$

The average is weighted by tonnage because the mill receives a blend of all delivered loads, so a load contributes to the overall grade in proportion to its tonnage. The cumulative state is then updated by $PROD \leftarrow PROD^{new}$ and $GRAD \leftarrow GRAD^{new}$. For a dump-bound or zero-load trip, both quantities remain unchanged.

10. Truck t_e becomes idle at the destination, and its origin for any subsequent trip is updated to $loc_{t_e} = dest(m)$. A further dispatch decision is considered only if the current time is less than H.

All trucks begin the shift at the parking lot, with $loc_t(0) = \text{park}$ and $PROD(0) = GRAD(0) = 0$. When a truck becomes idle, it takes no further trips if the current time is at least H or no candidate pile satisfies the operational and time-feasibility conditions. Other trucks and trips already underway continue. The simulation terminates once all truck processes have ended, and terminal performance is then evaluated.

3.8 Objective function

The objective combines a reward for contained metal delivered to the processing mill with penalties for production shortfall, grade departures from target, and exceedance of the upper mill-feed bound. Production above target is not penalized in itself; a capacity-violation penalty applies when mill feed exceeds its upper bound. Grade departures are penalized on both sides of the target to reflect the desired feed composition.

Let $PROD_H$ and $GRAD_H$ denote the final cumulative mill-feed tonnage and tonnage-weighted average grade, recorded after all trips initiated before H have completed.

Contained-metal term

The contained metal in the material delivered to the mill is the product of the final cumulative mill-feed tonnage and its tonnage-weighted average grade:

$$METAL_H = PROD_H \cdot GRAD_H. \tag{10}$$

When no material is delivered, $PROD_H = 0$ and $GRAD_H = 0$, so $METAL_H = 0$. Because $GRAD_H$ is maintained as the tonnage-weighted average grade of all material delivered to the mill, $PROD_H \cdot GRAD_H$ is equivalent to the total contained metal in the delivered mill feed. With mill-feed tonnage measured in tons and grades expressed as metal mass fractions, $METAL_H$ is measured in tons of contained metal. Since $GRAD_H$ is computed using the realized scenario grades $G_m(\omega)$, $METAL_H$ is scenario-dependent. This quantity measures contained metal before processing; metal prices and processing recovery are not modeled. The normalized contained-metal term is

$$\widehat{M}_H = \frac{METAL_H}{pt \cdot gt}, \tag{11}$$

where $pt \cdot gt$ is the contained metal delivered when the mill operates exactly at both targets. The normalized term is dimensionless and equals one at that operating point. It is assigned unit weight, while the penalty weights determine the relative importance of the remaining objective components.

Production-shortfall term

$$D_P^- = \max(0,\ pt - PROD_H), \tag{12}$$

the shortfall of mill-feed tonnage relative to the production target pt. Production above pt is not penalized by this term; exceedance of the upper mill-feed bound is addressed separately below.

Mill capacity-violation term

$$D_C = \max(0,\ PROD_H - up). \tag{13}$$

This term measures the amount by which delivered mill-feed tonnage exceeds the upper bound up.

Ore-grade deviation terms

The grade deviations are measured relative to the grade target gt and are defined only when material has been delivered to the mill:

$$\begin{aligned} D_G^- &= \max(0,\ gt - GRAD_H), \\ D_G^+ &= \max(0,\ GRAD_H - gt), \end{aligned} \qquad \text{if } PROD_H > 0. \tag{14}$$

If $PROD_H = 0$, both grade-deviation terms are set to zero. Here, D_G^- is the amount by which the average delivered grade falls below the target, and D_G^+ is the amount by which it exceeds the target.

Normalization

Each penalty is divided by the corresponding target-to-bound distance, yielding dimensionless quantities:

$$\widetilde{D}_P^- = \frac{D_P^-}{pt - lp}, \qquad \widetilde{D}_C = \frac{D_C}{up - pt}, \tag{15}$$

$$\widetilde{D}_G^- = \frac{D_G^-}{gt - lg}, \qquad \widetilde{D}_G^+ = \frac{D_G^+}{ug - gt}. \tag{16}$$

It is assumed that $lp < pt < up$ and $lg < gt < ug$ so that the penalty normalization is well-defined.

Objective

For a single simulation replication, let F denote the objective value calculated from the terminal quantities defined above:

$$F = \widehat{M}_H - \lambda_P \widetilde{D}_P^- - \lambda_G^- \widetilde{D}_G^- - \lambda_G^+ \widetilde{D}_G^+ - \lambda_C \widetilde{D}_C. \tag{17}$$

A higher value of F indicates better performance, combining contained-metal delivery with production, grade, and capacity control.

The goal is to find a scoring function $h \in \mathcal{H}$ that performs well on average across the modeled uncertainty. Formally, the problem is to maximize the expected objective,

$$\max_{h \in \mathcal{H}} \mathbb{E}_{\omega,\xi,A}[F(h)]\,, \tag{18}$$

where $\mathcal{H}$ is the class of scoring functions permitted by the rule representation and operation restrictions in Section 4.2, ω denotes the geological realization, ξ represents the stochastic operation-time outcomes, and A denotes the equipment-availability outcomes. The simulation replications used to estimate the expected objective are specified in Section 5.5. The resulting per-instance mean objective values are aggregated into training fitness as defined in Section 4.3. The relative weights $(\lambda_P, \lambda_G^-, \lambda_G^+, \lambda_C)$ control the trade-off between contained-metal delivery and the four penalty terms; their values are reported in Section 5.5.

Interpretation of the scale of F

When the mill operates exactly at both targets, $\widehat{M}_H = 1$ and all penalties vanish, giving the on-target reference value $F = 1$. Values above one are possible when $GRAD_H = gt$ and $pt < PROD_H \leq up$, in which case $F = PROD_H/pt > 1$. Grade deviations above and below the target are penalized separately, so deviations in opposite directions do not cancel when penalties are averaged across replications or instances. A rule can therefore deliver on-target contained metal in expectation, $\mathbb{E}[\widehat{M}_H] \approx 1$, while having a negative expected objective if its expected weighted penalties exceed its expected contained-metal reward.

3.9 Operational rules and feasibility conditions

The following rules are enforced through the simulation logic and govern the feasibility of dispatch decisions and state transitions.

3.9.1 Muck pile rules

Each muck pile represents an aggregation of one or more mining blocks. Extraction from each muck pile cannot exceed its initial tonnage:

$$0 \leq RTC_m(\tau_e) \leq TC_m \qquad \forall m \in \mathbb{M},\ \forall e \in \mathbb{E}. \tag{19}$$

A muck pile is dispatchable only if all of its predecessors have been fully depleted:

$$m \in \mathbb{M}^{feas}(\tau_e, t_e) \implies RTC_{m'}(\tau_e) = 0 \qquad \forall m' \in Pred(m). \tag{20}$$

3.9.2 Truck rules

As defined in Section 3.6, a truck can receive a new assignment only after completing its current service cycle and while the current time is before the dispatch horizon H.

For each dispatch event e, the realized payload satisfies

$$0 \leq Load_e \leq EC_{t_e}. \tag{21}$$

3.9.3 Shovel service rules

Shovel-to-muck-pile allocation is treated as a fixed input over the shift. In the current formulation, each muck pile is served by at most one shovel:

$$\sum_{s \in \mathbb{S}} W_{s,m} \leq 1 \qquad \forall m \in \mathbb{M}. \tag{22}$$

Each shovel serves at most one truck at a time. Trucks arriving at an occupied shovel wait in a first-come, first-served queue.

3.9.4 Destination service rules

The destination of each trip is determined by the selected muck pile through the mapping $dest(m)$ defined in Eq. (1). Each destination d has a finite service capacity CD_d, representing the number of trucks that can dump simultaneously. Trucks arriving when all CD_d service slots are occupied wait in a first-come, first-served queue. The destination-service time per truck is $DT_{t,d}$, and service begins as soon as a slot becomes available. The upper mill-feed bound up does not restrict the feasible muck-pile set at a dispatch event. Delivered tonnage may exceed up, with the excess penalized through the capacity-violation term in the objective.

3.9.5 Non-negativity

All normalized penalty terms are non-negative:

$$\widetilde{D}_P^- \geq 0, \qquad \widetilde{D}_G^- \geq 0, \qquad \widetilde{D}_G^+ \geq 0, \qquad \widetilde{D}_C \geq 0. \tag{23}$$

4 LLM-driven generation of dispatching rules

This section describes the framework for generating and selecting dispatching rules. Section 4.1 presents the overall workflow, and Section 4.2 defines the rule representation and input features. Sections 4.3 to 4.5 describe fitness evaluation and feedback, the instructions used to generate candidates, and the search procedure from initialization to final rule selection.

4.1 Framework overview

The framework uses the DES described in Section 3.7 to guide an evolutionary search over dispatching rules. The offline design process returns a single executable scoring function for subsequent dispatching decisions. Figure 1 summarizes the design process and the application of the selected rule.

The search maintains a population of candidate rules and an archive of high-performing rules found during the search. The archive is initialized with evaluated hand-crafted seed rules. At each generation, new candidates are requested from the LLM using one of four operator instructions: *explore*, *refine*, *simplify*, or *restructure*. Each request specifies the dispatching task and the allowed rule interface and asks for one executable scoring function. The operator determines whether the LLM designs a rule from scratch or revises an existing rule, as well as which archive rules and feedback are included in the prompt. Returned candidates are validated and checked for duplicates before evaluation. Candidates without recorded fitness are evaluated by DES on the training instances under uncertainty. Their simulated performance is used to calculate fitness and provide feedback for subsequent generations. The archive is updated after each generation, and the search continues until the generation limit is reached. The rule with the highest training fitness found during the search is then selected.

For an unseen instance, the dispatcher applies the operational and time-feasibility conditions of Section 3.6, computes the input features for each remaining candidate pile, and uses the selected rule

to calculate its priority score. The idle truck is assigned to the highest-scoring pile according to Eq. (5). The same rule is applied at successive dispatch events as the operational state changes, without further rule-design search or LLM calls.

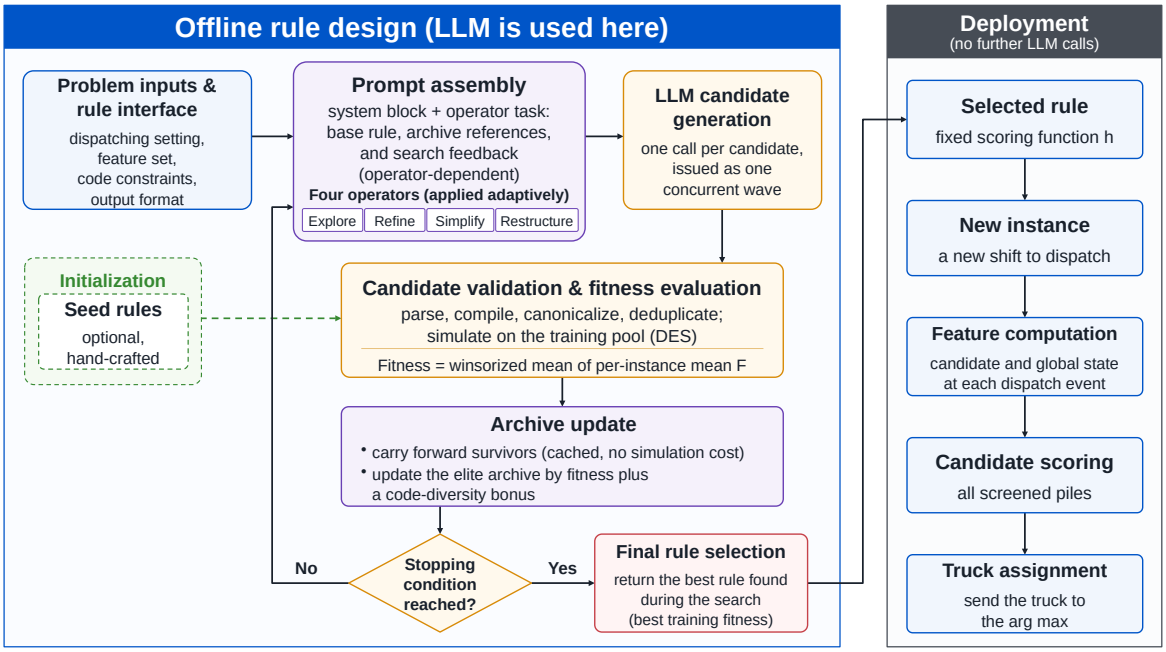

Figure 1: Workflow of the framework: offline rule design and deployment.

4.2 Rule representation and feature interface

A dispatching rule in this study is a scoring function h applied to every screened candidate pile at a dispatch event, as defined in Section 3.6. Rules are represented as short executable programs that read the feature vector $\phi(\tau_e, t_e, m)$ and return a scalar priority score; the truck is assigned to the highest-scoring candidate pile according to Eq. (5). The feature interface represents information available at dispatch time, including the estimated grade and remaining tonnage of a candidate pile, the state of its assigned shovel, nominal cycle information, shift progress, cumulative mill feed, and the delivered-grade state. Table 1 lists the available features in three groups. Pile-level features describe the candidate pile and the state of its assigned shovel; global features describe shift progress, cumulative production, and delivered grade; derived features combine these quantities into operational criteria such as throughput, production urgency, grade deviation, and projected overproduction. For each candidate pile, the rules use grade features based on its pre-shift estimate G^0_m. After each ore delivery, the cumulative delivered-grade state is updated using the realized grade of that load and used in subsequent assignments. The delivered-grade features are set to zero until the first positive delivery reaches the mill.

A candidate rule may combine these features with numerical constants using arithmetic operations, comparisons, conditional expressions, and numerical functions such as absolute value, minimum, maximum, and square root. To keep the generated rules self-contained and suitable for direct deployment, import statements, loops, class definitions, lambda expressions, exception handling, and persistent state across calls are prohibited. The code of each candidate rule is parsed and checked against these restrictions before simulation. Generated rules are also checked during evaluation for scoring errors and non-finite outputs. Such scores are treated as invalid, and shortest-queue dispatch is used as a fallback if all candidate scores at an event are invalid. These events are recorded during evaluation.

Table 1: Features available to the dispatching rules.

Feature	Definition	Meaning
Pile-level		
`G0_rel`	$(G_m^0 - gt)/gt$	Estimated pile grade relative to the mill grade target
`RTC_frac`	$RTC_m(\tau_e)/TC_m$	Remaining fraction of the pile
`load_frac`	$\min\{EC_{t_e}, RTC_m(\tau_e)\}/EC_{t_e}$	Fraction of truck capacity this trip would carry
`is_mill`	$\mathbf{1}[C_m = \text{ore}]$	Destination indicator
`queue_frac`	$QL_{s(m)}(\tau_e)/q_0$[a]	Physical queue at the assigned shovel
`committed_frac`	$\big(QL_{s(m)}(\tau_e) + EN_{s(m)}(\tau_e)\big)/q_0$[a]	Queued plus en-route trucks committed to the shovel
`cycle_frac`	$\big(\bar{RT} + \bar{LT} + \bar{HT} + \bar{DT}\big)/H$	Nominal full-cycle time of the candidate trip as a fraction of the shift
Global state		
`PROD_frac`	$PROD(\tau_e)/lp$	Delivered mill feed relative to the lower production bound
`time_frac`	$\max\{H - \tau_e, 0\}/H$	Remaining shift fraction
`PROD_gap_lower_frac`	$\max\{0,\ lp - PROD(\tau_e)\}/lp$	Production shortfall relative to the lower bound
`PROD_gap_upper_frac`	$\max\{0,\ PROD(\tau_e) - up\}/up$	Production excess beyond the upper bound
`GRAD_slack_lower_norm`	$(GRAD(\tau_e) - lg)/(ug - lg)$	Delivered-grade slack above the lower grade bound
`GRAD_slack_upper_norm`	$(ug - GRAD(\tau_e))/(ug - lg)$	Delivered-grade slack below the upper grade bound
`GRAD_dev_norm`	$(GRAD(\tau_e) - gt)/(ug - lg)$	Signed delivered-grade deviation from target, in band widths
Derived		
`throughput_rate`	`load_frac/cycle_frac`	Normalized nominal trip throughput
`queue_adjusted_throughput`	`load_frac` $/\big($`cycle_frac` $(1 +$ `queue_frac`$)\big)$	Normalized trip throughput discounted by shovel congestion
`grade_value`	`is_mill` $\cdot$ `G0_rel`	Estimated grade signal for mill-bound material
`mill_load_value`	`is_mill` $\cdot$ `load_frac`	Normalized load contribution for mill-bound material
`production_urgency`	`PROD_gap_lower_frac/time_frac`	Normalized production shortfall per unit remaining shift fraction
`grade_deficit`	$\max\{0,\ -$`GRAD_slack_lower_norm`$\}$	Amount by which the blend falls below the lower grade bound
`projected_overprod_frac`	$\max\{0,\ PROD(\tau_e) + \Delta_m - up\}/up$[b]	Projected excess beyond the upper bound if this trip is delivered

[a] q_0 is a fixed queue scale (Section 5.5), and $EN_s(\tau_e)$ is the number of trucks en route to shovel s at time τ_e.
[b] $\Delta_m =$ `is_mill` $\cdot \min\{EC_{t_e}, RTC_m(\tau_e)\}$ is the mill-feed tonnage the trip would deliver.

The complete admissible operation set and validation procedure are provided in the supplementary materials.

Features are normalized by fixed quantities associated with each instance, such as initial pile tonnage, truck capacity, shift length, and production and grade targets or bounds. Queue-related features use a fixed queue scale. This scaling gives the features comparable interpretations across instances with different absolute production levels and fleet sizes. The construction of the instance-specific targets and bounds is described in Section 5.1.

4.3 Fitness evaluation and feedback

A candidate rule h is evaluated on every instance of the training pool. Let $\bar{F}_j(h)$ denote the mean objective value over the replications of training instance j. The training fitness is the winsorized mean

$$F_{\text{train}}(h) = W_q\big(\bar{F}_1(h), \ldots, \bar{F}_J(h)\big)\,, \tag{24}$$

where J is the number of training instances and W_q denotes the winsorized mean. Winsorization replaces the lowest and highest fraction q of the per-instance values by the nearest remaining values before averaging. This limits the influence of unusually favorable or unfavorable instances on the search signal. The values of J and q are reported in Section 5.5. Before simulation, returned rules are validated and checked for duplicates. Duplicate candidates identified within a generation are evaluated only once, and previously evaluated code reuses its recorded fitness when it reappears. Candidates that pass validation and have no reusable fitness record are evaluated on the full training pool under the common simulation protocol. After each generation, the LLM receives aggregate feedback on search progress. This includes the best fitness found so far, the mean fitness of the current generation, and the current stagnation length. Rules included in a prompt are also accompanied by their fitness values. Per-instance performance values are not provided to the LLM.

4.4 Operator instructions

Each fresh LLM request is assigned one of four operator instructions: *explore*, *refine*, *simplify*, or *restructure*. The design follows the use of multiple prompt strategies in LLM-based evolutionary heuristic search (Liu et al., 2024). Explore generates a new rule without using an existing rule as its base, while the other three operators revise the best rule currently available in the archive.

Explore. The LLM is asked to design a new rule without a base or reference rule. The instruction encourages a distinct decision structure using the available feature semantics.

Refine. The LLM is asked to make a small, targeted modification to the base rule while preserving its main structure. Possible changes include adjusting the influence of a term, modifying production or grade control, or adding a conditional gate; a full rewrite is not permitted.

Simplify. The LLM is asked to retain the main behavior of the base rule while reducing its structural complexity. The instruction favors shorter expressions and permits at most one conditional branch.

Restructure. The LLM is asked to change the main decision logic of the base rule rather than making a local coefficient adjustment. For example, a rule centered on production pressure may be reorganized around grade state, congestion, or a state-dependent combination of criteria.

As an illustration, consider a compact base rule and illustrative outputs produced under the four operator instructions:

$$
\begin{aligned}
\text{base:}\quad & \texttt{mill_load_value} + 0.3\,\texttt{throughput_rate} - 0.5\,\texttt{queue_frac},\\
\textit{explore}:\quad & \texttt{throughput_rate}/(1+\texttt{committed_frac}) + \texttt{grade_deficit},\\
\textit{refine}:\quad & \texttt{mill_load_value} + 0.3\,\texttt{throughput_rate} - 0.8\,\texttt{queue_frac},\\
\textit{simplify}:\quad & \texttt{mill_load_value} - 0.5\,\texttt{queue_frac},\\
\textit{restructure}:\quad & \begin{cases} \texttt{production_urgency}\cdot\texttt{load_frac} - \texttt{queue_frac}, & \texttt{PROD_frac} < 1,\\ \texttt{grade_value} - \texttt{queue_frac}, & \text{otherwise.}\end{cases}
\end{aligned}
$$

These examples illustrate the intended scope of the four instructions; the LLM determines the actual modification returned by each request.

Every prompt contains a common system block describing the dispatching task, the output contract, and the feature interface of Section 4.2, followed by aggregate search feedback and the operator instruction. Additional prompt elements depend on the assigned operator, as summarized in Table 2. Refine, simplify, and restructure receive the current base rule. Refine and restructure additionally receive selected reference rules from the archive, together with their fitness values, and three fixed diagnostic notes. Explore receives neither a base rule nor reference rules.

An LLM generates text by sampling successive tokens from a probability distribution over possible outputs. The sampling temperature controls the concentration of this distribution: lower values place

Table 2: Composition of the generation prompt by operator.

Prompt element	Explore	Refine	Simplify	Restructure
System block (task, output contract, feature interface)	✓	✓	✓	✓
Aggregate search feedback	✓	✓	✓	✓
Base rule		✓	✓	✓
Reference rules with fitness values		✓		✓
Static diagnostic notes		✓		✓
Operator instruction	✓	✓	✓	✓
Output-format instruction	✓	✓	✓	✓

more probability on high-likelihood tokens and tend to produce more conservative outputs, whereas higher values increase output variability. In this framework, explore and restructure use slightly higher temperatures than refine and simplify because they are intended to permit broader changes in rule structure. The complete prompt text, including the four operator instructions and diagnostic notes, is provided in the supplementary materials.

4.5 Search procedure

This subsection describes one offline search run from initialization to final rule selection. Algorithm 1 summarizes the procedure, and Table 5 lists the experimental parameter settings.

Algorithm 1 One run of the LLM-driven evolutionary search.

Validate and evaluate the seed rules; initialize the archive with them
for each generation **do**
    Update the stagnation state from recent archive progress
    Carry forward the selected survivors from the archive and previous generation
    Assign an operator and temperature to each fresh request
    Issue the fresh LLM requests as one concurrent wave
    Validate, standardize, and deduplicate the returned rules
    Evaluate candidates without recorded fitness on the training pool
    Update the archive using fitness and code diversity
end for
return the rule with the highest training fitness found during the run

Before the first generation, the hand-crafted seed rules are passed through the same validation and fitness evaluation used for generated candidates, and the resulting rules initialize the archive. This initialization requires no LLM calls. The seed rules provide starting points that the revision operators can use, while the first generation itself consists entirely of LLM-generated candidates. Each generation contains a fixed number of candidate rules. From the second generation onward, survivors are selected from the highest-fitness rules in the archive and the preceding generation. A rule selected from both is carried forward only once. All positions not occupied by these distinct survivors are filled by fresh LLM requests. Overlap between the archive and previous-generation selections therefore increases the number of fresh requests needed to fill the generation. Survivors retain their recorded fitness and require no new simulation.

Fresh requests are distributed among the four operators of Section 4.4 according to a predefined schedule. Under normal progress, fixed proportions are assigned to *explore*, *refine*, *simplify*, and *restructure*. Search progress is monitored through the best fitness in the archive. If this best value improves by less than a specified threshold over a fixed number of recent generations, the run enters a stagnation state. The operator proportions then place greater emphasis on explore and restructure, and the sampling temperatures are increased moderately. When sufficient progress resumes, the normal proportions and temperatures are restored.

The archive preserves high-performing rules across generations while maintaining some structural diversity. Each rule is represented by the set of adjacent token pairs in its code, and the dissimilarity

between two rules is measured by their Jaccard distance, following the shingle-based representation of Broder (1997). For archive admission, each rule is scored using its training fitness together with a code-diversity bonus. The diversity bonus increases with the rule's dissimilarity from other archive candidates. It is scaled by the fitness range of the rules being compared so that its magnitude remains commensurate with the observed fitness differences. The highest-fitness rules are exempt from the diversity adjustment and retained directly.

After candidate evaluation and archive update, the next generation is formed and the process repeats until the fixed generation limit is reached. The archive may retain structurally diverse rules during the search, but final selection depends only on training fitness: the rule with the highest training fitness encountered during the run is returned. The returned rule is subsequently evaluated on unseen instances under the protocol of Section 5.5.

5 Experimental design

This section describes the computational study. Section 5.1 presents the instance generator, calibration procedure, and validation criteria. Section 5.2 specifies the geological realizations and instance pools. Section 5.3 introduces the automated rule-design benchmark and specifies its configuration. Section 5.4 presents the predefined dispatching rules used as additional baselines. Section 5.5 defines the training and test protocols, computational budgets, statistical procedures, and configuration of the LLM search.

5.1 Instance generation and validation

The computational study uses a synthetic instance generator calibrated from an existing open-pit copper-mine case study developed from operational data collected from two open-pit mines equipped with fleet management systems. The case provides representative truck and shovel capacities, equipment availabilities, loading and dumping times, haul distances, production requirements, and ore-grade information for blasted muck piles. The distributional form used to generate synthetic pile grades is specified in Section 5.2. The present study extends this case-calibrated setting into a generator that produces multiple training and test instances for automated rule design and out-of-sample evaluation.

An instance specifies the muck piles and their tonnages, estimated grades and precedence relations, the shovel allocation, the truck fleet, travel times, and mill targets. The generator contains four modules. The topology module creates a layered precedence graph and assigns each shovel to a distinct initially exposed muck pile. The geometry module locates the piles and derives the travel-time tables. The grade module generates estimated pile grades and associated grade distributions. The fleet module specifies truck counts, payload classes, and service times. The case study provides reference values for equipment and operating parameters. The generator introduces variation in mine layout, fleet composition, material characteristics, and production conditions to produce diverse instances.

Table 3 summarizes the principal instance parameters. The truck payload classes, shift length, equipment availabilities, and service times are based on the case study. Standard instances contain 6 to 12 total muck piles, 2 to 4 shovels, and 12 to 24 trucks. Large instances contain 20 to 30 total muck piles, 5 to 8 shovels, and 40 to 60 trucks. Each shovel remains assigned to one initially exposed muck pile throughout the shift, while trucks are dynamically assigned among the shovel-served piles at successive dispatch events. Apart from the ranges governing muck-pile count, shovel count, and fleet size, the same sampling mechanisms are used for both instance scales.

Mill-feed tonnage and ore-grade targets are calibrated separately for each instance so that they represent comparable operating requirements across different fleet and mine scales. Production capacity is estimated before simulation as the minimum of three quantities: the tonnage that the shovels serving ore piles can load during the shift, the tonnage that the truck fleet can transport from those piles, and the total ore tonnage available at the shovel-served piles. The estimate uses nominal equipment rates

Table 3: Instance parameters.

Parameter	Standard	Large
Total muck piles	6 to 12	20 to 30
Shovels	2 to 4	5 to 8
Trucks	12 to 24	40 to 60
Pile tonnage	20,000 to 90,000 tons, per pile	
Precedence layers	2 to 4	
Truck payload classes	150 and 200 tons	
Shift length H	10 hours	
Operation-time variability CV	0.10, shared across operations	
Availabilities SA_s, TA_t	0.85 per shift	
Nominal loading time	0.04 to 0.07 hours, per instance	
Nominal dumping time (mill; dump)	0.015 to 0.025; 0.018 to 0.028 hours	
Destination service capacity	mill 1, dump 2	
Mill-feed tonnage target	$pt = 0.70\times$ estimated capacity; $lp, up = pt(1 \mp 0.10)$	
Mill-feed ore-grade target and bounds	0.50-quantile target; 0.30/0.70 bounds; minimum $\pm 2\%$ of gt	
Pools (train; test)	30; 30	30; 30

and available tonnage and does not depend on the dispatching rule. The production target is set at $pt = 0.70$ of this estimated capacity, with lower and upper bounds $lp = 0.9\,pt$ and $up = 1.1\,pt$.

The grade target gt is the tonnage-weighted blend of the shovel-served ore piles at the 0.50 quantile of their grade distributions. The lower and upper grade bounds are obtained from the corresponding blends at the 0.30 and 0.70 quantiles. Each bound is required to lie at least 2% of gt from the target, which prevents the normalization in Section 3.8 from being driven by an excessively narrow grade band. The resulting targets therefore adapt to the grade characteristics of each instance while preserving a comparable interpretation of normalized deviations.

The production-utilization fraction and the grade-band settings were selected in preliminary calibration using classical dispatching rules. The calibration sought settings under which production and grade requirements were neither almost always satisfied nor almost always violated, while keeping the normalized objective components on comparable scales. Because production and grade targets are calibrated to each instance, the normalized state variables of Section 4.2 retain comparable interpretations across instances with different absolute production levels and fleet sizes.

The generator uses three grade regimes: *high-grade*, in which most piles have estimated grades above the pre-shift ore/waste classification threshold; *mixed*, with a more balanced combination of ore and waste; and *near-cutoff*, with estimated pile grades concentrated close to this same classification threshold, or cutoff grade. Each training and test pool contains ten instances from each regime to provide balanced coverage during rule design and evaluation.

Each candidate instance passes structural and behavioral checks before entering a pool. The structural screen verifies precedence-graph integrity and requires at least $\max(3, |\mathbb{S}|)$ initially exposed piles, at least two shovel-served ore piles, total mill-destined tonnage of at least $2\,lp$, and an estimated-grade spread of at least 10^{-4} among the shovel-served ore piles. The behavioral screen evaluates shortest-queue and mill-first dispatching using the 0.30- and 0.70-quantile grade scenarios, with three operational replications per scenario. An instance is retained if at least one rule achieves both a mean objective above -4.5 and mean mill production above $0.75\,lp$. The same screening procedure is applied to the standard and large-instance pools.

5.2 Geological realizations and instance pools

For the synthetic instances described in Section 5.1, each geological realization specifies one possible set of grades for all muck piles. The scenario-generation procedure operates directly at the pile level: the generator specifies a marginal grade distribution for each pile and uses a Gaussian copula to sample spatially dependent grades according to the pile locations. The resulting grade vectors are stored

before dispatching simulation and reused during rule evaluation. This subsection describes the marginal distributions, spatial-dependence model, alternative percentile scenarios, and instance pools.

To generate instances with different estimated grades and uncertainty levels using a common parametric specification, the grade module represents each pile m by a normal distribution with location $\mu_m = G_m^0$ and scale σ_m, symmetrically truncated to $[\mu_m - 3\sigma_m, \mu_m + 3\sigma_m]$. This construction centers uncertainty on each pile's pre-shift grade estimate, allows the spread to vary between piles, and bounds deviations from the estimate. Each instance stores seven quantiles at levels 0.05, 0.10, 0.30, 0.50, 0.70, 0.90, and 0.95. These quantiles are calculated from the pile-specific distributions and stored to encode them. Under this symmetric construction, μ_m is recovered from the median and σ_m from the 0.05–0.95 quantile spread using the corresponding unit-scale quantiles. The recovered parameters are checked by reconstructing all seven stored quantiles. The resulting continuous quantile function Q_m is used to generate realization grades.

A Gaussian copula is used to sample pile grades jointly while preserving their prescribed marginal distributions (Nelsen, 2006). The sampling uses auxiliary Gaussian variables, with their dependence determined by the distances between piles:

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \Sigma), \qquad \Sigma_{ij} = \exp(-d_{ij}/r), \qquad u_m = \Phi(z_m), \qquad G_m(\omega) = Q_m(u_m), \tag{25}$$

where $\mathbf{z}$ is an auxiliary vector of correlated standard normal variables with correlation matrix Σ, d_{ij} is the distance between piles i and j, and Φ is the standard normal distribution function. The correlation range r is set to one half of the largest pairwise pile distance in the instance. Scaling r with the layout size gives the generated instances a common relative spatial-dependence scale. Each simulated value z_m is transformed into a quantile level $u_m = \Phi(z_m)$, and $Q_m(u_m)$ gives the corresponding grade of pile m. This construction produces stronger dependence between the grades of nearby piles while preserving each pile's marginal grade distribution.

For each instance, $K = 15$ equally weighted realizations are generated using a fixed random seed and labeled R01 to R15. The generated grades are stored with the instance and reused throughout the computational study. The realization subsets and operational replications used for training and testing are specified in Section 5.5.

For comparison, an alternative training representation uses three synchronized percentile scenarios, P30, P50, and P70, corresponding to the percentile levels used to define the ore-grade target and bounds in Section 5.1. In each scenario, every pile is assigned the same quantile level of its own grade distribution; for example, P30 assigns $Q_m(0.30)$ to each pile m. The three scenarios therefore provide a compact set of below-median, median, and above-median grade conditions. By contrast, the Gaussian-copula realizations used in the main training and test evaluations assign continuously varying, spatially correlated quantile levels across piles. Piles within the same realization may therefore occupy different parts of their marginal distributions, with realized grades extending below P30 or above P70.

The standard training and test pools contain 30 instances each and are disjoint. They are assembled through an ordered scan over generator seeds, with the training quotas filled first. The large training and test pools likewise contain 30 instances each and use seed ranges separate from the standard pools and from each other. All four pools follow the grade-regime quotas and screening criteria of Section 5.1, and their geological realizations are generated using the same procedure. For an operating-mine study, pile-level grade scenarios can be obtained by aggregating grades from geostatistical simulations of the orebody (Goovaerts, 1997; Dimitrakopoulos and Yao, 2023). These scenarios can be supplied directly to the dispatching simulation, without imposing the marginal distribution family or Gaussian-copula construction used for the synthetic instances.

5.3 Automated rule-design benchmark

GP is used as the automated rule-design benchmark for the proposed LLM search. The LLM search and GP use the same feature interface, training instances, simulation environment, geological realizations, and operational replications.

The GP implementation maintains a population of candidate dispatching rules. Each candidate, referred to as an individual, is represented by an expression tree. The leaves contain input features from Table 1 or numerical constants, and the internal nodes specify operations that combine these values. The available operations include arithmetic, protected division, and a conditional operator. Protected division handles near-zero denominators, while the conditional operator returns the value of one of two expressions depending on whether its condition is positive. Like an LLM-generated rule, the resulting expression maps the features of a candidate pile to a scalar priority score. Figure 2 illustrates this representation using an example expression tree. The complete sets of operations and constants are provided in the supplementary materials. Each GP rule is applied in the DES, and its training performance is measured by the winsorized mean of per-instance objective values defined in Eq. (24). A parsimony penalty of 10^{-5} per tree node is subtracted from this value to discourage unnecessarily large expressions. The adjusted fitness guides evolutionary selection and the choice of the final rule. The runtime checks described in Section 4.2 also apply to GP-generated rules.

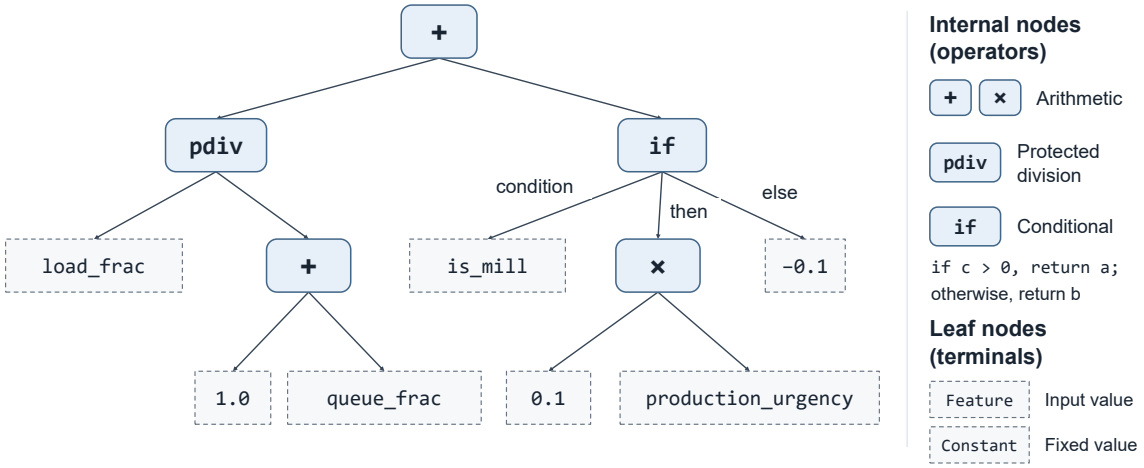

Figure 2: Example GP expression tree for a dispatching rule.

The GP search starts from an initial population of expression-tree rules, each evaluated in the DES on the training pool using the adjusted fitness defined above. The search then proceeds generation by generation: tournament selection chooses parent rules by comparing fitness within randomly sampled groups, subtree crossover exchanges parts of parent expressions, subtree mutation replaces part of an expression with a newly generated subtree, and elitism carries the best rule forward to the next generation. Each run uses a population of 300 rules for 70 generations, with a maximum tree depth of 6 to limit expression complexity. These selection and variation operators, together with the control of expression complexity, are consistent with established tree-based GP approaches to automated scheduling heuristic design (Koza, 1992; Burke et al., 2013; Branke et al., 2015). The full parameter settings are reported in Table 5. At termination, the rule with the highest adjusted training fitness in the final population is returned.

5.4 Baseline rules

This subsection introduces fourteen manually designed dispatching rules as baselines for evaluating the automatically generated rules. The set includes classical and single-attribute rules, the three

seed rules used in the LLM search, and a tuned DISPATCH-style rule. Table 4 summarizes their decision criteria. All rules are evaluated in the same DES under the dispatching feasibility conditions of Section 3.6 and the test protocol of Section 5.5. The three seed rules address queue avoidance, queue-adjusted throughput, and production and grade deficits, respectively. They initialize the LLM search and are also evaluated separately as dispatching rules. Their standalone performance provides a reference for assessing how the final LLM-generated rules compare with the initial rules.

The tuned DISPATCH-style rule, denoted `industry_proxy_tuned`, is inspired by the production-plan and congestion considerations of the DISPATCH system introduced in Section 2.1 (White and Olson, 1986; Alarie and Gamache, 2002; Moradi Afrapoli et al., 2019). It combines these considerations in a single scoring function. The cost of each feasible trip includes its nominal cycle time and weighted estimates of waiting time at the shovel and destination. To account for production requirements, this cost is reduced for mill-bound trips when cumulative mill feed falls behind a linear schedule toward the shift production target. The reduction increases with the production-plan shortfall, giving greater priority to ore delivery when production is behind schedule. The weights $(w_{\text{wait}}, w_{\text{dest}}, w_{\text{plan}})$ control the shovel-wait, destination-wait, and plan-lag terms, respectively, and the truck is assigned to the feasible pile with the lowest cost. The three weights were calibrated on the standard training pool through a four-round coarse-to-fine grid search. The selected setting, $(w_{\text{wait}}, w_{\text{dest}}, w_{\text{plan}}) = (5, 1, 16)$, was then held fixed for evaluation on both test pools.

Table 4: Baseline rules.

Baseline	Definition
Classical dispatch rules	
`random`	Uniform random choice among screened candidate piles
`fixed_assignment`	Static truck allocation proportional to shovel service rates
`shortest_queue`	Fewest trucks at the candidate pile's assigned shovel
`sptf`	Shortest nominal cycle plus estimated shovel waiting time
`needy_shovel`	Largest shortfall from the target share of truck assignments
`mill_first_shortest_queue`	Ore piles before waste; ties broken by shortest queue
`grade_aware`	Greedy rule guided by the delivered-grade deviation
Single-attribute rules	
`grade_greedy`	Highest estimated grade first
`max_load`	Largest feasible load first
`max_remaining`	Largest remaining tonnage first
Seed rules and tuned DISPATCH-style rule	
`seed_shortest_queue`	Normalized shortest-queue rule (seed rule 1)
`seed_qat`	Queue-adjusted throughput (seed rule 2)
`seed_deficit_aware`	Production- and grade-deficit responsive composite (seed rule 3)
`industry_proxy_tuned`	Tuned DISPATCH-style rule, weights $(5, 1, 16)$

5.5 Protocol, budgets, and statistical methodology

This subsection specifies the training, validation, and test protocols, together with the performance measures, computational-cost accounting, and statistical methodology. The main configurations of the LLM search and GP are summarized in Table 5.

For each training-pool instance, realizations R01 to R08 are used for candidate evaluation during the search, while R09 to R15 are reserved for preliminary configuration comparisons. The reserved realizations are used to compare the Gaussian-copula and percentile-scenario training representations introduced in Section 5.2. Based on this comparison, the main configuration uses Gaussian-copula realizations.

For the main comparison, the LLM search and GP are each run independently 30 times on the standard training pool. Candidate rules are evaluated on realizations R01 to R08, with three operational replications per realization. Each rule is therefore evaluated through 24 simulations per instance and

$30 \times 24 = 720$ simulations across the training pool. Simulation outcomes are averaged within each instance, and the resulting instance means are aggregated using the winsorized mean of Eq. (24), with $q = 0.1$. GP additionally applies the parsimony penalty described in Section 5.3. The main experiments use $\lambda_P = \lambda_G^- = \lambda_G^+ = \lambda_C = 1$, so the four normalized penalty terms receive equal coefficients.

Table 5: Configuration of the LLM search and GP.

Setting	Value
LLM search	
Generations per run	30
Population per generation	16 candidate rules
Survivors (from generation 2)	up to 4: top 2 of the archive, top 2 of the previous generation
Fresh LLM requests	all non-survivor slots (12 or more); 16 in the first generation
Seed rules	3
Winsorized training fitness	$q = 0.1$ over $J = 30$ training instances
Queue scale q_0	5 trucks
Archive	
Archive size	8 rules; top 3 retained directly by training fitness
Diversity measure	Jaccard distance between sets of adjacent token pairs
Diversity bonus weight	0.05× fitness range of the rules considered
Operators and LLM	
LLM	Gemini 3 Flash (`gemini-3-flash-preview`)
Reference rules shown	5
Operator proportions	explore 25 %, refine 35 %, simplify 10 %, restructure 30 % under stagnation: 30 %, 20 %, 10 %, 40 %
Stagnation rule	best archive fitness improves by < 0.005 over 3 generations
Temperatures	explore, restructure: 1.05 (1.10 under stagnation) refine, simplify: 0.95 (1.00 under stagnation)
GP	
Population; generations	300; 70
Selection	tournament, size 3
Crossover; mutation	subtree, rate 0.8; subtree, rate 0.2
Elitism	1
Initialization	ramped half-and-half
Depth limit	6
Parsimony penalty	10^{-5} per node, subtracted from training fitness

Each run returns one rule, which is applied unchanged to both test pools. Testing uses all 15 geological realizations (R01 to R15) stored for each test instance, with five operational replications per realization, giving 75 simulations per rule on each test instance. The mean objective value and mean objective components are recorded for subsequent analyses. Operational randomness is generated using fixed seeds for equipment availability and operation durations. The same seed set is reused across rules and realizations within each instance. This use of common random numbers is intended to reduce simulation noise in comparisons between rules.

Following reference-based performance reporting in scheduling benchmarks (Taillard, 1993), test performance is measured by the mean gap to a fixed best-observed value on each instance. For a rule h, let $\bar{F}_j(h)$ denote its mean objective over the 75 test replications of instance j. Let F_j^* denote the highest such mean attained on instance j by the rules included in the initial evaluation of that test pool. The reference values remain fixed when additional rules are evaluated. The mean gap is

$$\mathrm{Gap}(h) = \frac{1}{N_{\mathrm{test}}} \sum_{j=1}^{N_{\mathrm{test}}} \left(F_j^* - \bar{F}_j(h) \right),$$

where $N_{\mathrm{test}} = 30$ is the number of instances in each test pool. Lower gap values indicate better performance. The gap is expressed in F units rather than as a percentage because objective values can be negative or close to zero.

Computational cost is measured by the elapsed time from the start to the end of each search run (wall-clock time) and the recorded number of fitness evaluations. A full rule evaluation covers all training instances and their simulation replications. The LLM count increases whenever a rule is evaluated through simulation; reuse of a stored fitness value does not increase this count. The GP count records the number of individuals that require a fitness value. When duplicate GP rules occur within the same generation, the rule is evaluated once on the training pool and the resulting fitness is assigned to all corresponding individuals. Each individual is nevertheless included in the recorded evaluation count.

Performance across independent runs is summarized using the pool-level mean gap of each returned rule. For each run, the returned rule is evaluated on all 30 test instances, and its mean gap across these instances is calculated. The reported mean and standard deviation are then computed across the independent runs. The standard deviation therefore reflects variation between independently generated rules. Statistical comparisons use the test instance as the unit of analysis. For each learned method and each test instance, $\bar{F}_j(h)$ is averaged over the rules returned by its independent runs, yielding one performance value for that method on the instance. A baseline rule contributes its own $\bar{F}_j$ on the same instance. Each pairwise comparison therefore consists of 30 paired instance-level values, which are evaluated using a two-sided Wilcoxon signed-rank test. Mean performance differences and win/loss counts across the 30 test instances are reported alongside the tests. Holm correction is applied to groups of related pairwise comparisons within each test pool. The comparison groups are specified alongside the corresponding results.

6 Results and discussion

This section evaluates the performance of the generated dispatching rules and the computational effort required to obtain them. Section 6.1 presents the main comparison on the standard test pool, including rule quality, variation across independent runs, and computational cost. Section 6.2 examines search progress, and Section 6.3 evaluates generalization to larger instances and retraining at that scale. Section 6.4 examines the objective components and sensitivity to alternative weights. Section 6.5 compares training and search configurations, and Section 6.6 analyzes the selected LLM rule and discusses the operational implications of the results.

6.1 Main comparison on the standard test pool

This subsection compares the LLM search with GP and the fourteen manually designed dispatching rules on the standard test pool. Table 6 reports the mean gaps defined in Section 5.5. For the learned methods, results are shown as mean ± standard deviation across 30 independent runs. The column Δ reports each method's mean gap minus that of the LLM search, and W counts the test instances on which the LLM search performs better than the method in that row. The p values correspond to paired Wilcoxon comparisons with the LLM search, with Holm correction across the fourteen baseline comparisons.

As shown in Table 6, the LLM search achieves a numerically lower mean gap than GP, 0.257 versus 0.290, although the difference is not statistically significant ($p = 0.237$). It performs better than GP on 19 of the 30 test instances. The LLM search also shows lower variation between independent runs, with standard deviations of 0.049 for the LLM search and 0.085 for GP.

Both learned methods achieve substantially lower mean gaps than all fourteen manually designed baselines. The LLM search outperforms every baseline with statistical significance after Holm correction ($p < 0.001$). The best-performing baseline, `grade_aware`, has a mean gap of 0.833, which exceeds the LLM search's gap by 0.575. The `industry_proxy_tuned` has a gap of 0.993, and the LLM search performs better on 29 of the 30 instances. Even the largest mean gap among the 30 LLM runs, 0.389, is below that of the best-performing baseline. All 30 LLM-generated rules therefore outperform every

Table 6: Results on the standard test pool.

Method	Gap	Δ	W	p
LLM search (30 runs)	0.257 ± 0.049	–	–	–
GP (30 runs)	0.290 ± 0.085	+0.032	19	0.237
`grade_aware`	0.833	+0.575	22	< 0.001
`seed_deficit_aware`	0.873	+0.615	30	< 0.001
`mill_first_shortest_queue`	0.991	+0.733	30	< 0.001
`industry_proxy_tuned`	0.993	+0.736	29	< 0.001
`random`	1.381	+1.123	28	< 0.001
`needy_shovel`	1.447	+1.190	29	< 0.001
`fixed_assignment`	1.489	+1.232	28	< 0.001
`seed_qat`	1.610	+1.352	29	< 0.001
`sptf`	1.664	+1.406	29	< 0.001
`shortest_queue`	1.731	+1.474	30	< 0.001
`seed_shortest_queue`	1.731	+1.474	30	< 0.001
`max_remaining`	2.431	+2.173	30	< 0.001
`grade_greedy`	3.383	+3.126	30	< 0.001
`max_load`	3.771	+3.514	30	< 0.001

manually designed baseline in mean performance across this test pool. The ranking of the baselines also reveals differences between dispatching criteria. Rules that respond to delivered grade or combine production and grade criteria perform better than shortest-queue and shortest-trip rules on this pool. Nevertheless, the best-performing seed rule, `seed_deficit_aware`, has a mean gap of 0.873, which exceeds that of the LLM search by 0.615. The final rules thus achieve a substantial improvement over the initial hand-crafted rules. The contribution of seed initialization to the search is examined in Section 6.5, while Section 6.4 relates the performance differences to production and grade outcomes.

The LLM search also requires substantially less computation time than GP. Table 7 reports average computational costs over 30 independent runs on a laptop with an Intel Core i9-12900H processor and 32 GB of memory. The mean wall-clock time is 10.7 minutes for the LLM search and 2.5 hours for GP, so the LLM search requires approximately 7.1% of GP's recorded computation time. The mean recorded evaluation count is 408 per run for the LLM search and 17,843 per run for GP. The ratio of the LLM count to the GP count is approximately 2.3%, under the counting conventions defined in Section 5.5. The LLM search therefore achieves a numerically lower average test gap and lower run-to-run variability with substantially shorter computation time. Performance on larger unseen instances is examined in Section 6.3.

Table 7: Measured cost per run.

Method	Recorded evaluation count	LLM calls	Tokens in / out (millions)	Wall-clock time
LLM search	408	421	1.38 / 0.14	10.7 min
GP	17,843	–	–	2.5 h

6.2 Search behavior

This subsection examines how the LLM search and GP improve rule quality during training. Figure 3 shows the best training fitness at each generation, summarized by the median and interquartile range across the 30 independent runs of each method. The dashed line extends the LLM search's final median level beyond generation 30 for comparison with the longer GP search. The GP curve uses the adjusted training fitness defined in Section 5.3.

The LLM search reaches a median best fitness of −0.76 in its first generation after seed initialization. Its median curve improves rapidly during the early generations and then approaches a final level of −0.47. By generation 27, the median best fitness is within 0.005 of its final value. GP starts from

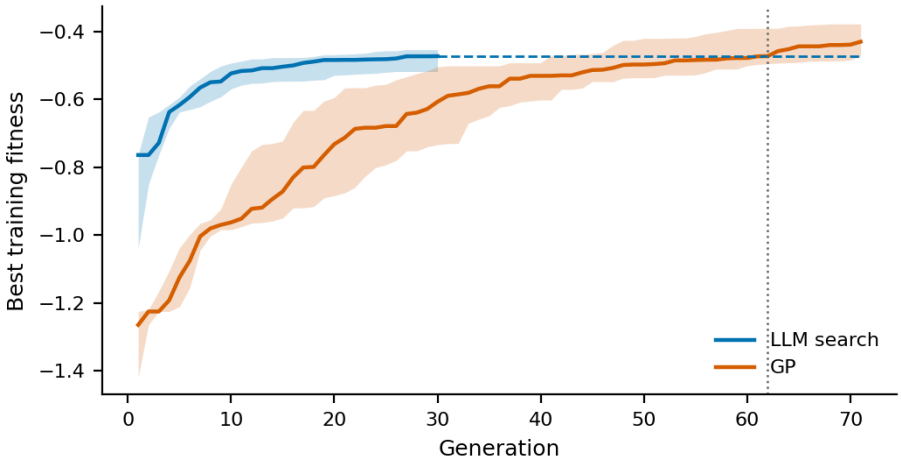

Figure 3: Best training fitness by generation.

randomly generated expression trees with an initial median best fitness of −1.27 and improves more gradually. It exceeds the LLM search's final median level at generation 62, after approximately 15,600 recorded individual evaluations, and finishes at −0.43. Despite this higher final training fitness, GP does not achieve a lower mean gap on the standard test pool (Table 6). The LLM search thus obtains effective rules within its shorter search, with the lower computational cost reported in Table 7.

Failures in LLM candidate generation are infrequent. Across the 30 main runs, an average of 1.3 of the 421 calls per run fail to produce a parseable rule, with no more than three such failures in any run. Each failed call is followed by a replacement request.

6.3 Generalization to larger instances and retraining

This subsection evaluates whether rules learned on standard instances remain effective on larger unseen instances. The large test instances are generated independently of the standard training instances using the same sampling mechanisms for grades and operating parameters, but with more muck piles, shovels, and trucks, as specified in Table 3. This comparison examines whether rules can be reused at a larger operating scale under the same distributional assumptions. The same 30 LLM-generated rules and 30 GP-generated rules evaluated in Section 6.1 are applied unchanged to this pool. Table 8 reports their performance alongside the fourteen manually designed baselines and five exploratory LLM runs trained directly on the large training pool. The columns follow the definitions in Table 6, with all comparisons made against the standard-trained LLM search. Holm correction is applied across the fourteen baseline comparisons within the large test pool.

The standard-trained LLM search achieves a mean gap of 0.389, compared with 0.718 for GP, and performs better on 29 of the 30 large test instances ($p < 0.001$). The difference in mean gap is 0.329, compared with 0.032 on the standard test pool. The performance advantage of the LLM-generated rules is therefore more pronounced when they are applied beyond the training scale. Variation across independent runs also remains lower for the LLM search, with a standard deviation of 0.212 compared with 0.470 for GP.

Both learned methods achieve lower mean gaps than every manually designed baseline. The standard-trained LLM search outperforms all fourteen baselines with statistical significance after Holm correction ($p < 0.001$). The best-performing baseline on this pool is `industry_proxy_tuned`, the tuned DISPATCH-style rule, with a gap of 0.895. The LLM search reduces this gap by 0.506 and performs better on 24 of the 30 instances. The baseline ranking also changes with instance scale:

Table 8: Results on the large-instance test pool.

Method	Gap	Δ	W	p
LLM search (30 runs, standard-trained)	0.389 ± 0.212	–	–	–
GP (30 runs, standard-trained)	0.718 ± 0.470	+0.329	29	< 0.001
LLM search (5 runs, large-trained)	0.216 ± 0.021	−0.173	4	< 0.001
`industry_proxy_tuned`	0.895	+0.506	24	< 0.001
`seed_deficit_aware`	1.651	+1.262	30	< 0.001
`mill_first_shortest_queue`	1.907	+1.518	30	< 0.001
`needy_shovel`	1.953	+1.564	30	< 0.001
`random`	2.082	+1.693	30	< 0.001
`fixed_assignment`	2.120	+1.731	30	< 0.001
`sptf`	2.231	+1.842	30	< 0.001
`seed_qat`	3.420	+3.031	30	< 0.001
`grade_aware`	3.967	+3.578	29	< 0.001
`shortest_queue`	4.224	+3.835	30	< 0.001
`seed_shortest_queue`	4.224	+3.835	30	< 0.001
`max_remaining`	6.215	+5.826	30	< 0.001
`max_load`	8.913	+8.523	30	< 0.001
`grade_greedy`	9.629	+9.240	30	< 0.001

`grade_aware`, the best-performing baseline on the standard pool, has a gap of 3.967 on the large pool. The LLM-generated rules remain ahead of all manually designed baselines despite this change in their relative performance.

Training directly on the large instances further improves the LLM-generated rules. The five large-trained runs achieve a mean gap of 0.216 with a standard deviation of 0.021, compared with 0.389 for the standard-trained LLM search. The large-trained rules perform better on 26 of the 30 instances ($p < 0.001$). The standard-trained LLM rules already outperform GP and all manually designed baselines when applied directly to the larger unseen instances, without retraining or further LLM calls. These results demonstrate strong generalization of the LLM-designed rules across instance scales.

6.4 Objective decomposition and weight sensitivity

This subsection examines the objective components underlying the performance differences in Sections 6.1 and 6.3 and the sensitivity of these comparisons to the penalty weights. The comparisons concern the combined objective F, which rewards contained-metal delivery while penalizing mill-feed shortfall, ore-grade deviations, and overfeed. Table 9 reports normalized contained metal and the four penalty terms for the LLM search, GP, and two manually designed baselines: `grade_aware` and `industry_proxy_tuned`. These are the best-performing baselines under unit weights on the standard and large test pools, respectively. The entries are means of the normalized objective components in Eq. (17), averaged over test replications and instances and, for the learned methods, over 30 independent runs. Under unit weights, the mean objective equals the mean contained-metal reward minus the four mean penalty terms.

On the standard pool, the LLM search and GP deliver nearly identical mean normalized contained metal, at 1.018 and 1.017, respectively. Adding the below-target and above-target grade penalties gives $0.479 + 0.447 = 0.926$ for the LLM search and $0.514 + 0.497 = 1.011$ for GP. The reduction in combined grade penalties outweighs the LLM search's slightly larger production-shortfall and overfeed penalties and accounts for its numerical advantage in mean F under unit weights. On the large pool, the LLM search achieves a higher mean metal reward and lower mean values of all four penalties than GP. Its advantage at this scale therefore extends across production, grade, and capacity control.

The two manually designed baselines exhibit different trade-offs. The `grade_aware` rule has relatively small grade and overfeed penalties, but its production-shortfall penalty reaches 1.179 on the standard pool and 4.264 on the large pool. The `industry_proxy_tuned` rule delivers the most normalized contained metal among the four methods and has smaller production shortfalls than the learned

Table 9: Objective decomposition.

Method	$\widehat{M}_H$	$\widetilde{D}_P^-$	$\widetilde{D}_C$	$\widetilde{D}_G^-$	$\widetilde{D}_G^+$
Standard test pool					
LLM search	1.018	0.485	0.223	0.479	0.447
GP	1.017	0.452	0.203	0.514	0.497
`grade_aware`	0.907	1.179	0.080	0.445	0.395
`industry_proxy_tuned`	1.079	0.311	0.536	0.840	0.744
Large test pool					
LLM search	0.932	0.963	0.042	0.252	0.333
GP	0.924	1.062	0.091	0.312	0.444
`grade_aware`	0.578	4.264	0.009	0.235	0.306
`industry_proxy_tuned`	0.982	0.656	0.159	0.545	0.785

methods, together with larger grade and overfeed penalties. Relative to this baseline, the LLM search achieves 94.3% of its mean normalized contained metal on the standard pool, while reducing the combined grade penalties by 41.5% and the overfeed penalty by 58.3%. On the large pool, it achieves 94.9% of the baseline's mean normalized contained metal, with reductions of 56.1% in combined grade penalties and 73.9% in the overfeed penalty. The LLM-generated rules therefore retain approximately 95% of the baseline's mean normalized contained-metal delivery while substantially improving ore-grade and overfeed control.

The weight analysis examines how these trade-offs affect performance comparisons when the relative importance of the penalties changes. For each rule trained under the unit-weight setting, we keep the rule unchanged and recompute its objective value under alternative penalty weights using the recorded objective components; no retraining is performed. Each penalty weight is halved and doubled separately, and all four weights are also halved and doubled together. This gives ten alternative settings in addition to the unit-weight case.

In Table 10, Δ denotes the difference in mean F between the LLM search and GP, and p reports the corresponding paired Wilcoxon test. The p values are Holm-adjusted across the eleven weight settings within each test pool. The margin column reports the difference in mean F between the LLM search and the best-performing baseline among the fourteen rules under each weight setting. The baseline is identified by its mean performance across the test pool. Positive values of Δ and margin favor the LLM search.

Table 10: Sensitivity to objective weights.

	Standard pool			Large pool		
Weights	Δ	p	margin	Δ	p	margin
all $\lambda = 1$	+0.0323	1.000	+0.575	+0.3292	< 0.001	+0.506
$\lambda_P \times 0.5$	+0.0486	0.704	+0.228	+0.2794	< 0.001	+0.659
$\lambda_P \times 2$	−0.0004	1.000	+0.563	+0.4287	< 0.001	+0.199
$\lambda_C \times 0.5$	+0.0424	0.427	+0.580	+0.3044	< 0.001	+0.447
$\lambda_C \times 2$	+0.0119	1.000	+0.432	+0.3789	< 0.001	+0.624
$\lambda_G^- \times 0.5$	+0.0147	1.000	+0.523	+0.2989	< 0.001	+0.359
$\lambda_G^- \times 2$	+0.0673	0.427	+0.541	+0.3897	< 0.001	+0.799
$\lambda_G^+ \times 0.5$	+0.0073	1.000	+0.363	+0.2733	< 0.001	+0.280
$\lambda_G^+ \times 2$	+0.0821	0.192	+0.523	+0.4411	< 0.001	+0.958
all $\times 0.5$	+0.0163	1.000	+0.306	+0.1684	< 0.001	+0.228
all $\times 2$	+0.0642	1.000	+1.040	+0.6508	< 0.001	+1.062

On the standard pool, the LLM search has a numerically higher mean F than GP in ten of the eleven weight settings in Table 10, but none of the differences is statistically significant after Holm correction. On the large pool, it outperforms GP under every tested setting, with mean-F advantages from 0.168 to 0.651 and all comparisons significant after correction ($p < 0.001$). Its higher mean contained-metal reward and lower mean value of each penalty explain this consistent ranking. For these

evaluated rules, the mean-F advantage is preserved under any nonnegative penalty weights when the contained-metal reward retains unit weight. The LLM search also maintains a higher mean F than every baseline on both pools under all tested settings, with a minimum margin of 0.199. When all four penalty weights are halved, the minimum advantages over the baselines remain 0.306 on the standard pool and 0.228 on the large pool.

To examine larger reductions than those in Table 10, we consider a common value for all four penalty weights. The contained-metal reward retains unit weight. We use the unrounded mean components underlying Table 9 to calculate the common weight at which the LLM search and `industry_proxy_tuned` have equal mean F. This occurs at approximately 0.077 on the standard pool and 0.091 on the large pool. Above the respective thresholds, the LLM search has the higher mean F. At a common penalty weight of 0.1, the LLM search therefore still outperforms this baseline in mean F on both pools.

6.5 Ablation variants

This subsection uses ablation comparisons to examine how removing search components or changing training and model settings affects the quality of the generated rules. The comparisons cover training scenarios, the choice of LLM, search control, seed initialization, and training scale. Table 11 defines the variants and reports their numbers of independent runs. The training-scenario comparison is also conducted for GP to examine whether the observed differences extend beyond LLM-based rule generation. All returned rules are evaluated on both test pools under the protocol of Section 5.5.

Table 11: Ablation variants.

Variant	Main configuration	Variant setting	Runs
Percentile-scenario training	Eight copula realizations	P30, P50, P70	30
DeepSeek model	Gemini 3 Flash	DeepSeek V4 Flash	10
Simplified search control	Adaptive operator settings; archive diversity bonus	Fixed operator settings; no diversity bonus	10
No seed rules	Three seed rules	No seed rules	10
Trained on large instances	Standard training pool	Large training pool	5
GP, percentile-scenario training	Eight copula realizations	P30, P50, P70	5

The variants in Table 11 examine alternative training data and search settings. For both the LLM search and GP, percentile-scenario training replaces the eight Gaussian-copula realizations with P30, P50, and P70, with three operational replications per scenario. Each candidate rule therefore requires 9 training simulations per instance in this configuration, compared with 24 in the main configuration. Both configurations use the same test protocol. The DeepSeek variant replaces Gemini 3 Flash with DeepSeek V4 Flash. The simplified search-control variant fixes operator proportions and sampling temperatures at their normal-progress values and removes the archive diversity bonus, while retaining the four operator instructions and performance feedback. The no-seed variant omits the three hand-crafted seed rules, and the large-instance training variant replaces the standard training pool with the large training pool.

Table 12 reports mean gaps and standard deviations across the independent runs of each configuration. For every variant, Δ is its mean gap minus that of the corresponding main configuration, so positive values indicate worse performance. The paired Wilcoxon tests follow Section 5.5, with Holm correction across the six variant-to-main comparisons within each test pool.

The results support the realization-based training configuration selected in the preliminary comparisons of Section 5.5. On the standard pool, the LLM rules trained on percentile scenarios have a mean gap of 0.318, compared with 0.257 for the main configuration, a difference of 0.060 ($p = 0.033$). On the large pool, the difference increases to 0.272, with the realization-trained rules performing better on all 30 instances ($p < 0.001$). The GP comparison follows the same direction: percentile-scenario training increases the mean gap by 0.039 on the standard pool, where the difference is not significant

Table 12: Results of the ablation variants.

	Standard test pool			Large test pool		
Variant	Gap	Δ	p	Gap	Δ	p
LLM search, main configuration	0.257 ± 0.049	–	–	0.389 ± 0.212	–	–
Percentile-scenario training	0.318 ± 0.058	+0.060	0.033	0.661 ± 0.241	+0.272	< 0.001
DeepSeek model	0.348 ± 0.055	+0.091	< 0.001	0.816 ± 0.325	+0.427	< 0.001
Simplified search control	0.248 ± 0.041	−0.009	0.594	0.305 ± 0.157	−0.084	< 0.001
No seed rules	0.230 ± 0.025	−0.028	0.594	0.380 ± 0.240	−0.010	0.952
Trained on large instances	0.292 ± 0.059	+0.035	0.594	0.216 ± 0.021	−0.173	< 0.001
GP, main configuration	0.290 ± 0.085	–	–	0.718 ± 0.470	–	–
GP, percentile-scenario training	0.329 ± 0.172	+0.039	0.111	0.957 ± 0.418	+0.239	< 0.001

($p = 0.111$), and by 0.239 on the large pool ($p < 0.001$). The realization-based configuration combines spatially varying grade outcomes with more simulation replications per candidate. Its performance advantage is particularly evident when the resulting rules are applied to larger unseen instances.

Replacing Gemini 3 Flash with DeepSeek V4 Flash (`deepseek-v4-flash`) increases the mean gap by 0.091 on the standard pool and by 0.427 on the large pool, with both differences significant ($p < 0.001$). On the large pool, the Gemini 3 Flash configuration performs better on 29 of the 30 instances. One of the ten DeepSeek runs also returns a rule that produces invalid scores and triggers the shortest-queue fallback described in Section 4.2 on both test pools.

The search-control comparison examines stagnation-responsive operator settings and diversity-based archive selection. These mechanisms encourage broader candidate generation when progress slows and favor structurally different rules in the archive. The simplified configuration instead uses fixed operator settings and fitness-based archive selection. It reaches a mean gap of 0.248 on the standard pool, compared with 0.257 for the main configuration; the difference is not significant ($p = 0.594$). On the large pool, its gap is 0.305, an improvement of 0.084, with better performance on 27 of the 30 instances ($p < 0.001$). Both configurations retain the same LLM-based rule generation and performance-feedback process. These results show that effective rules can be generated with fixed operator settings and fitness-based archive selection. The simplified configuration also achieves better performance on the large test pool.

Removing the three seed rules yields mean gaps of 0.230 on the standard pool and 0.380 on the large pool. Both values are numerically lower than those of the seeded search, although neither difference is statistically significant ($p = 0.594$ and $p = 0.952$, respectively). Variation across runs is also smaller for the unseeded search on the standard pool, with a standard deviation of 0.025 compared with 0.049. Seed initialization gives refine, simplify, and restructure evaluated rules to revise from the first generation, whereas without seeds these operators initially fall back to explore while the archive is empty. This difference is reflected in the training curves of Figure 4: the seeded search begins with a fitness advantage of approximately 0.28, although the unseeded search catches up by generation 5 and finishes slightly higher. The main configuration retains the seeds to provide an immediate starting point for revision-based search, at a cost of three initial rule evaluations. These results show that effective rules can also be developed without seed initialization.

The five LLM rules trained on large instances in Section 6.3 are also applied to the standard test pool to examine generalization to smaller instances. They achieve a mean gap of 0.292, compared with 0.257 for the standard-trained LLM rules; the difference is not statistically significant ($p = 0.594$). Their mean gap remains below that of every manually designed baseline in Table 6. Together with the results on larger instances, this comparison demonstrates strong generalization across the two tested instance scales: rules trained at either scale remain effective when applied directly at the other, without further search or LLM calls.

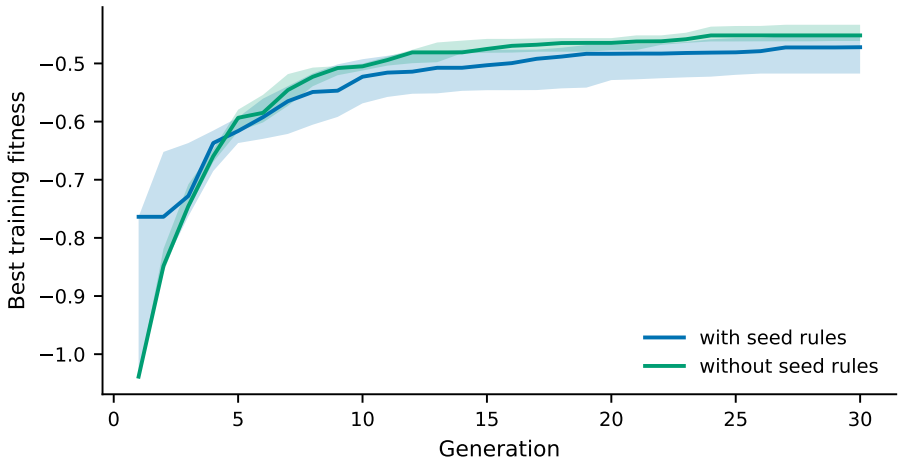

Figure 4: Best training fitness of the LLM search with and without seed rules.

6.6 Rule analysis and operational implications

This subsection examines rule selection and the structure of the generated dispatching rules, followed by the implications of offline rule design for real-time dispatching. Independent search runs can generate different rules, so selecting a rule for use requires a criterion that is informative about performance on unseen instances. Across the 30 runs of each method, the Spearman rank correlation between training fitness and mean gap on the standard test pool is -0.75 for the LLM search and -0.90 for GP ($p < 0.001$ for both). Higher training fitness is associated with lower test gaps, supporting its use to select among the rules generated by independent runs.

Figure 5 illustrates the LLM rule with the highest training fitness among the 30 main runs. The example shows how a single generated rule can combine several operational criteria and respond to different production states.

The rule distinguishes mill-bound and waste-bound trips and changes their relative priority as production progresses. For mill-bound trips, it combines a production catch-up term, ore-grade correction, and penalties for projected overfeed. The grade-correction term favors above-target material when the delivered blend is below target and below-target material when the blend is above target. Congestion-adjusted throughput accounts for the load, nominal cycle time, and trucks committed to the shovel. Waste haulage receives greater priority once mill feed reaches a specified production level. The example therefore demonstrates how a generated rule can represent different operating situations within a single explicit scoring function. The explicit scoring rule also provides a basis for engineering review: its scoring terms and branching conditions allow an engineer to trace how production, ore-grade, and congestion information influence an assignment. The logic can be checked against operating requirements, and proposed changes can be evaluated in the same simulation environment without requiring the engineer to reproduce each score calculation manually.

The separation between rule design and rule execution also has practical value. The computation times in Table 7 show that offline rule design takes substantially less time with the LLM search than with GP. Once a rule is selected, dispatching requires feature computation and evaluation of its arithmetic and conditional expressions; no LLM call or optimization search is required at a dispatch event. The coefficients remain fixed, while the scores respond to changes in the operating state. For an operating-mine application, representative local conditions can be used to construct the training instances and simulation environment for offline rule design, after which candidate rules can be assessed

and selected before deployment. The generalization results in Section 6.3 show that selected rules can remain effective on unseen instances within the tested setting, including instances with larger fleets.

```
def score(features):
prod = features["PROD_frac"]
dev = features["GRAD_dev_norm"]
is_mill = features["is_mill"]
congestion = 1.0 + features["committed_frac"] * 1.9
throughput = features["load_frac"] / max(features["cycle_frac"] * congestion, 1e-9)
mill_target = 1.11
mill_limit = 1.25
if is_mill > 0.5:
grade_corrective = -1.0 * features["G0_rel"] * dev * (25.0 + 15.0 * abs(dev))
metal_incentive = features["G0_rel"] * max(0.0, mill_limit - prod)
if prod < 0.85:
base_pull = 2.8 * (mill_target - prod) + features["production_urgency"]
score_val = (base_pull + metal_incentive + grade_corrective) * throughput
elif prod < mill_limit:
margin = mill_limit - prod
risk_brake = features["projected_overprod_frac"] * 350.0
quality_focus = grade_corrective * (1.5 + abs(dev))
score_val = (quality_focus + metal_incentive * margin) * throughput - risk_brake
else:
score_val = (-1500.0 * features["projected_overprod_frac"]
- 800.0 * features["PROD_gap_upper_frac"] + grade_corrective)
else:
if prod < 0.75:
waste_urgency = -18.0
elif prod < mill_target:
waste_urgency = (prod - mill_target) * 12.0
else:
waste_urgency = (5.0 + (prod - mill_target) * 55.0
+ (1.0 - features["RTC_frac"]) * 2.5)
score_val = waste_urgency * throughput
return float(score_val)
```

Figure 5: The LLM rule with the best training fitness

7 Conclusions and future work

This paper presents an LLM-driven evolutionary framework for the offline design of truck-dispatching rules under geological and operational uncertainty. The objective rewards contained metal delivered to the mill and penalizes production shortfall, ore-grade deviations, and overfeed. Each candidate rule is evaluated by simulation on the training instances, and its per-instance mean objective values are aggregated into a fitness value that guides subsequent rule generation and revision. Each offline search run returns the rule with the highest training fitness for direct application to unseen instances without further search or LLM calls.

The computational study evaluates rules generated by the LLM search on standard and large-instance test pools, using GP as an established automated rule-design benchmark. On the standard test pool, the LLM search achieves numerically better average test performance than GP and shows lower variation across independent runs. It also requires substantially less offline search effort: its recorded evaluation count and mean computation time are approximately 2.3% and 7.1% of those of GP, respectively. On the large-instance test pool, the standard-trained LLM rules significantly outperform the standard-trained GP rules and all manually designed baselines. The large-trained LLM rules also perform effectively when applied to the standard test pool. Together, the results from applying standard-trained rules to large instances and large-trained rules to standard instances demonstrate strong generalization of the LLM-generated rules across the two tested instance scales.

The objective decomposition shows that the LLM-generated rules maintain high metal delivery alongside effective ore-grade and overfeed control. Their large-instance advantage over GP and their mean-performance advantages over the manually designed baselines persist across the tested penalty weights. The ablation experiments further show that training and search configuration choices can materially affect the quality of the generated rules.

As an illustration, the LLM rule with the highest training fitness is examined in detail. The rule combines production progress, ore-grade feedback, congestion, and other operational information within a compact scoring function, demonstrating that the search can generate rules that respond to multiple aspects of the dispatching state. Once a rule is selected, real-time dispatching requires only feature computation and score evaluation at each decision event. The framework therefore separates computationally intensive rule design from inexpensive online execution.

Future research could investigate more informative performance feedback and targeted generation instructions to further improve rule quality and search efficiency. The framework could also be extended to richer operating settings, including multi-shift operations, shovel relocation, stockpile management, multi-metal blending, and additional sources of equipment uncertainty such as within-shift breakdown and repair processes. A further direction is to connect the rule-design framework with fleet management systems and short-term production planning, so that dispatching targets can be set from production plans and subsequent planning decisions can be adjusted based on realized dispatching performance. These extensions would broaden the framework from single-shift truck dispatching toward more integrated mine planning and operational decision support.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work, the authors used ChatGPT (OpenAI) to support language editing and manuscript organization. The authors reviewed and edited the content as needed and take full responsibility for the content of the article.

References

Stéphane Alarie and Michel Gamache. Overview of solution strategies used in truck dispatching systems for open pit mines. International Journal of Surface Mining, Reclamation and Environment, 16(1):59–76, 2002.

Christian Both and Roussos Dimitrakopoulos. Joint stochastic short-term production scheduling and fleet management optimization for mining complexes. Optimization and Engineering, 21(4):1717–1743, 2020.

Juergen Branke, Su Nguyen, Christoph W Pickardt, and Mengjie Zhang. Automated design of production scheduling heuristics: A review. IEEE Transactions on Evolutionary Computation, 20(1):110–124, 2015.

Andrei Z. Broder. On the resemblance and containment of documents. In Proceedings of the Compression and Complexity of Sequences, pages 21–29. IEEE, 1997.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society, 64 (12):1695–1724, 2013.

İbrahim Oğuz Çetinkaya, İ. Esra Büyüktahtakın, Parshin Shojaee, and Chandan K. Reddy. Discovering heuristics with Large Language Models (LLMs) for mixed-integer programs: Single-machine scheduling. Computers & Operations Research, 186:107325, 2026. doi: 10.1016/j.cor.2025.107325.

Patarawan Chaowasakoo, Heikki Seppälä, Heikki Koivo, and Quan Zhou. Digitalization of mine operations: Scenarios to benefit in real-time truck dispatching. International Journal of Mining Science and Technology, 27(2):229–236, 2017. doi: 10.1016/j.ijmst.2017.01.007.

Shikun Chen. Coupling-aware destruction with large-language-model guidance for energy-efficient blocking hybrid flow shop scheduling. Computers & Operations Research, 196:107645, 2026. doi: 10.1016/j.cor.2026.107645.

Joao Pedro de Carvalho and Roussos Dimitrakopoulos. Integrating production planning with truck-dispatching decisions through reinforcement learning while managing uncertainty. Minerals, 11(6):587, 2021.

Joao Pedro de Carvalho and Roussos Dimitrakopoulos. Integrating short-term stochastic production planning updating with mining fleet management in industrial mining complexes: an actor-critic reinforcement learning approach. Applied Intelligence, 53:23179–23202, 2023. doi: 10.1007/s10489-023-04774-3.

João Pedro de Carvalho and Roussos Dimitrakopoulos. Simultaneous shovel allocation and grade control decisions for short-term production planning of industrial mining complexes – an actor-critic approach. International Journal of Mining, Reclamation and Environment, 38(1):53–78, 2024. doi: 10.1080/17480930.2023.2247196.

Roussos Dimitrakopoulos and Lingqing Yao. High-order spatial stochastic models. In Encyclopedia of Mathematical Geosciences, pages 605–613. Springer, Cham, 2023. doi: 10.1007/978-3-030-85040-1_16.

Cristiane Ferreira, Gonçalo Figueira, and Pedro Amorim. Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. Omega, 111:102643, 2022. doi: 10.1016/j.omega.2022.102643.

Pierre Goovaerts. Geostatistics for Natural Resources Evaluation. Oxford University Press, New York, 1997.

Arman Hazrathosseini and Ali Moradi Afrapoli. Transition to intelligent fleet management systems in open pit mines: A critical review on application of reinforcement-learning-based systems. Mining Technology, 133(1): 50–73, 2024.

Jin Huang, Yue Teng, Qihao Liu, Liang Gao, Xinyu Li, Chunjiang Zhang, and Guoqing Xu. Leveraging large language models for efficient scheduling in Human-Robot collaborative flexible manufacturing systems. npj Advanced Manufacturing, 2:47, 2025. doi: 10.1038/s44334-025-00061-w.

Sungbum Jun and Seokcheon Lee. Learning dispatching rules for single machine scheduling with dynamic arrivals based on decision trees and feature construction. International Journal of Production Research, 59 (9):2838–2856, 2021. doi: 10.1080/00207543.2020.1741716.

John RR Koza. Genetic programming: On the programming of computers by means of natural selection, volume 1. MIT press, 1992.

Ashish Kumar, Roussos Dimitrakopoulos, and Marco Maulen. Adaptive self-learning mechanisms for updating short-term production decisions in an industrial mining complex. Journal of Intelligent Manufacturing, 31(7): 1795–1811, 2020.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: towards efficient automatic algorithm design using large language model. In Proceedings of the 41st International Conference on Machine Learning, ICML'24. JMLR.org, 2024.

Jingyu Luo, Mario Vanhoucke, José Coelho, and Jie Song. Automated design of heuristics for resource-constrained project scheduling problem via regression algorithms. Annals of Operations Research, 2026. doi: 10.1007/s10479-026-07196-9.

Hossein Mirzaei-Nasirabad, Mehrnaz Mohtasham, Hooman Askari-Nasab, and Behrooz Alizadeh. An optimization model for the real-time truck dispatching problem in open-pit mining operations. Optimization and Engineering, 24:2449–2473, 2023. doi: 10.1007/s11081-022-09780-x.

Ali Moradi Afrapoli, Mohammad Tabesh, and Hooman Askari-Nasab. A multiple objective transportation problem approach to dynamic truck dispatching in surface mines. European Journal of Operational Research, 276(1):331–342, 2019.

Mohan Munirathinam and Jon C Yingling. A review of computer-based truck dispatching strategies for surface mining operations. International Journal of Surface Mining and Reclamation, 8(1):1–15, 1994.

Roger B. Nelsen. An Introduction to Copulas. Springer Series in Statistics. Springer, New York, 2 edition, 2006.

Su Nguyen, Yi Mei, and Mengjie Zhang. Genetic programming for production scheduling: a survey with a unified framework. Complex & Intelligent Systems, 3(1):41–66, 2017.

Roberto Noriega, Yashar Pourrahimian, and Hooman Askari-Nasab. Deep reinforcement learning based real-time open-pit mining truck dispatching system. Computers & Operations Research, 173:106815, 2025.

Junyoung Park, Jaehyeong Chun, Sang Hun Kim, Youngkook Kim, and Jinkyoo Park. Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. International Journal of Production Research, 59(11):3360–3377, 2021. doi: 10.1080/00207543.2020.1870013.

Junhao Qiu, Haoyang Zhuang, Fei Liu, Jianjun Liu, and Qingfu Zhang. EvoDR: Evolving dispatching rules via large language model for dynamic flexible assembly flow shop scheduling, 2026. URL https://arxiv.org/abs/2601.15738v2. arXiv preprint arXiv:2601.15738, version 2.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. Nature, 625(7995):468–475, 2024.

Yuning Shi, Xiaolei Ma, Peng Zhang, and Huabo Lu. A dynamic truck scheduling framework for open-pit mine production under equipment fault uncertainty. Expert Systems with Applications, 299:130254, 2026. doi: 10.1016/j.eswa.2025.130254.

Amanda Smith, Jeff Linderoth, and James Luedtke. Optimization-based dispatching policies for open-pit mining. Optimization and Engineering, 22:1347–1387, 2021. doi: 10.1007/s11081-021-09628-w.

Eric Taillard. Benchmarks for basic scheduling problems. European Journal of Operational Research, 64(2): 278–285, 1993.

Niki Van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. IEEE Transactions on Evolutionary Computation, 29(2):331–345, 2024.

Martha E. Villalba Matamoros and Roussos Dimitrakopoulos. Stochastic short-term mine production schedule accounting for fleet allocation, operational considerations and blending restrictions. European Journal of Operational Research, 255(3):911–921, 2016. doi: 10.1016/j.ejor.2016.05.050.

J. W. White and J. P. Olson. Computer-based dispatching in mines with concurrent operating objectives. Mining Engineering, 38(11):1045–1054, 1986.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. Advances in neural information processing systems, 37:43571–43608, 2024.

Yunhui Zeng, Zijun Liao, Yuanzhi Dai, Rong Wang, Xiu Li, and Bo Yuan. Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism, 2022. arXiv preprint arXiv:2201.00548.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In Advances in Neural Information Processing Systems, volume 33, 2020.

Fangfang Zhang, Yi Mei, Su Nguyen, and Mengjie Zhang. Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. IEEE Transactions on Evolutionary Computation, 28 (1):147–167, 2024.