Capability-Bounded Code Evolution with Large Language Models
A Cross-Family Study of Adaptive Heuristic Search

by

Fabio Cozzuto

A dissertation submitted to the
Department of Computer Science
in conformity with the requirements for
the degree of Master of Science

Bishop's University
Canada
August 2026

Abstract

Large language models (LLMs) can now write and revise executable code, which raises a concrete question for heuristic optimization: can an iterative language-model loop generate useful heuristic programs under controlled evaluation, and what limits appear when benchmarks and baselines become more demanding? The study addresses that question through a unified experimental framework in which models propose deterministic Python heuristics, receive task-specific validator feedback, and are compared under replicated conditions across increasingly realistic problem families. The empirical progression begins with simple adversarial games, moves to the symmetric traveling salesperson problem (TSP), the asymmetric traveling salesperson problem (ATSP), and the capacitated vehicle routing problem (CVRP), and then studies CVRP solver evolution on Uchoa X-series benchmark instances from the Capacitated Vehicle Routing Problem Library (CVRPLIB). The retained evidence supports a capability-bounded interpretation. The loop can produce measurable behavioral adaptation in games and some held-out routing improvements. The same evidence shows that novelty spikes are often unreliable, replay effects are family-dependent, and no single replay recipe dominates TSP, ATSP, and CVRP. In later phases, modular operator search rediscovered recognizable operator families without validating a reusable operator, and adaptive portfolio control improved over a weak LLM selector while remaining behind the strongest fixed selector. Real-world CVRP solver evolution preserved full held-out feasibility and improved over weaker baselines while remaining behind Clarke–Wright savings. A matched-budget CVRP control study showed that replay-aware incumbent-preserving search had lower penalized gap than both a memoryless learned control and a bounded direct-generation-plus-repair control, mainly by avoiding feasibility collapse. An external PyVRP solver calibration based on hybrid genetic search (HGS) ideas produced lower gaps than both Clarke–Wright and the evolved solvers. Finally, a budget-matched factorial study found no replay, failure, or compression condition that reliably improved on the no-replay control. The resulting contribution is a measurement framework for distinguishing useful adaptation from code churn, search budget from mechanism value, and bounded heuristic improvement from unsupported claims of general solver discovery.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr. Russell Butler, for his guidance, encouragement, and thoughtful feedback throughout this research.

I thank the Computer Science Department and the professors who taught me during the program, especially Dr. Rachid Hedjam, whose support extended both inside and outside the classroom. Their courses, guidance, and expectations helped shape the background needed to complete this work.

I am grateful to Bishop's University and Jimmy Couturier for their academic, administrative, and financial support.

Scholarships from NSERC (CGRS M) and the Arbour Foundation provided continuity during this project and made it possible for me to carry the work forward.

Finally, I thank my family and friends for their encouragement, patience, and practical help throughout my Master's studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The design of effective heuristics remains a central practical problem in computer science. Routing, scheduling, resource allocation, and related optimization tasks are often too expensive to solve exactly at the scales required in practice. In these settings, practitioners rely on hand-crafted heuristics, metaheuristics, and solver portfolios whose quality depends on human insight, empirical tuning, and domain knowledge [8, 13, 26]. The question of how to automate parts of heuristic design has remained active across operations research, machine learning, and evolutionary computation.

Recent progress in large language models (LLMs) changes the form of this question. Modern code models can synthesize non-trivial executable programs from natural-language specifications, solve benchmark programming tasks, and improve candidate programs through repeated sampling and automated testing [3, 9, 17, 18, 19, 35]. For optimization, one plausible generated program is not enough. The stronger question is whether an outer loop that proposes code, validates it, retains useful variants, and reuses evidence from previous attempts can produce reliable held-out improvements under controlled conditions.

The completed work studies that question empirically. It begins with simple adversarial games, then moves to benchmark routing families: the symmetric traveling salesperson problem (TSP), the asymmetric traveling salesperson problem (ATSP), and the capacitated vehicle routing problem (CVRP). The later experiments include bounded whole-solver evolution on Uchoa X-series instances from the Capacitated Vehicle Routing Problem Library (CVRPLIB), matched-budget learned controls, a hybrid genetic search (HGS)-style CVRP calibration baseline, and a crossed model-strength study. This progression tests whether apparent progress survives stronger validators, stronger baselines, and more realistic feasibility constraints.

1.1 Problem Statement and Research Question

Here, LLM-guided code evolution is treated as a measurable adaptive search loop with no assumption of autonomous algorithm invention. A model receives a constrained programming interface and task feedback, proposes a deterministic Python heuristic, and receives structured information about the heuristic's behavior and performance. The loop repeats over multiple epochs, storing code variants, validator signals, and archive summaries that may be replayed into later prompts.

The central research question is:

> Under what conditions does an LLM-guided code-evolution loop generate useful heuristic adaptation, and when are apparent gains better explained by search budget, benchmark headroom, validator pressure, or baseline strength?

This question is examined through narrower empirical questions. The study asks whether replay-aware memory improves held-out performance relative to no replay and alternative replay rules, whether effects observed in simple games transfer to TSP, ATSP, and CVRP benchmarks, and whether the loop can produce reusable operators, adaptive controllers, or feasible whole solvers under stricter validation. It also asks how much apparent progress remains once comparisons use matched budgets, held-out instances, and stronger classical baselines.

1.2 Objectives and Contributions

To answer these empirical questions, the thesis proceeds by addressing four connected aspects of the problem: formalizing the evaluation loop, testing it across different benchmark families, isolating the role of replay from other sources of apparent improvement, and reporting the evidence in a reproducible form.

The first objective is to formalize LLM-guided code evolution as a controlled experimental pipeline with deterministic validators, archived runs, held-out panels, and explicit baselines. The second objective is to evaluate that pipeline across benchmark families whose structure and difficulty differ materially. The third objective is to separate replay-specific effects from effects due to search budget, model strength, and benchmark headroom. The fourth objective is to report both positive and negative findings in a form that supports reproducible evaluation instead of best-run anecdotes.

The resulting contribution is a cross-family empirical account of LLM-guided heuristic code evolution. It includes a common framework for studying executable heuristic search, a set of replicated studies across simple games and routing benchmarks, a matched-budget analysis of replay-aware CVRP solver evolution, a calibration against a modern hybrid genetic search (HGS)-style CVRP solver, and a

synthesis that connects statistical results to representative generated-code artifacts and failure modes.

1.3 Empirical Hypotheses

The hypotheses below are empirical claims tested by the completed campaigns. They are not assumptions built into the method.

H1: Adaptive search under validator feedback. When a benchmark family contains reachable heuristic headroom and the interface remains executable under repeated sampling, an iterative LLM-guided loop can generate non-trivial behavioral diversity and sometimes improve held-out quality over simpler generation.

H2: Replay must justify its added machinery. A replay or archive mechanism has independent value only when it improves held-out performance relative to simpler controls or alternative replay rules evaluated under the same protocol.

H3: Gains shrink under stronger evaluators. Apparent advantages of evolved code should weaken as validator pressure rises, feasibility constraints tighten, and classical baselines consume more of the available benchmark headroom.

H4: Transfer is partial. Search components discovered on one family may transfer across related problems. The strength of that transfer should depend on the target domain's structure rather than on a task-independent notion of general algorithmic discovery.

These hypotheses permit mixed or negative outcomes. Under controlled evaluation, null results are informative because they separate mechanism effects from artifacts of sampling budget, weak baselines, or favorable benchmark choice.

1.4 Scope

The dissertation studies executable heuristic programs. Direct solution construction is outside the scope of the reported experiments. In the early game phases, the generated artifact is policy code for a two-player environment. In the routing phases, the generated artifact is TSP, ATSP, or CVRP heuristic code evaluated against benchmark reference costs. In the later CVRP phases, the artifact is a bounded solver that must return complete capacity-feasible routes on held-out Uchoa X-series CVRPLIB instances. The project does not claim state-of-the-art routing performance or a general theory of algorithm discovery. Its purpose is to measure where the loop adapts, where it fails, and which controls are necessary before a mechanism-specific claim is credible. The routing benchmarks use the TSPLIB benchmark library for TSP and ATSP and CVRPLIB for CVRP.

Family	Benchmark basis	Evolved artifact	Primary held-out metric
Simple games	synthetic adversarial environments	deterministic policy code	win rate or score margin
TSP	TSPLIB symmetric instances	tour-construction or improvement heuristic code	relative optimality gap
ATSP	TSPLIB asymmetric instances	transfer-tested heuristic code	relative optimality gap
CVRP	CVRPLIB, especially Uchoa X-series instances	route-construction or whole-solver code	penalized gap with feasibility pressure

Table 1.1: Problem families covered by the dissertation and the corresponding evaluation focus.

1.5 Thesis Statement

Across simple games, TSP, ATSP, and real-world CVRP, LLM-guided code evolution can generate diverse executable heuristic programs and sometimes preserve held-out improvements. The evidence also shows that this capability is bounded. Performance depends on validator pressure, baseline strength, benchmark headroom, and task-specific interaction structure. Replay-specific mechanisms can be useful under some controls. The experiments do not support a universal replay advantage or practical dominance over mature human-designed optimization heuristics.

1.6 Organization of the Dissertation

Chapter 2 consolidates the mathematical background and related work on heuristics, benchmark routing problems, machine learning for combinatorial optimization, LLM-based code generation, evaluator-guided program search, replay, and diversity mechanisms. Chapter 3 defines the conceptual framework, notation, experimental protocol, statistical reporting, and artifact policy. Chapter 4 combines the experimental setup, results, and analysis for all retained campaigns. Chapter 5 assesses the hypotheses, states the answer to the main research question, and presents limitations and future work. Appendix A provides representative evolved artifacts and failure cases.

Chapter 2

Background and Literature Review

Chapter 2 consolidates the conceptual background and related work used throughout the thesis. It defines the terms needed to read the empirical chapters, then situates the work within heuristics, hyper-heuristics, algorithm selection, benchmark routing, machine learning for combinatorial optimization, LLM-based code generation, evaluator-guided program search, and replay mechanisms.

2.1 Core Concepts

Combinatorial optimization studies problems in which a feasible solution must be selected from a large discrete set. In this thesis, a problem family consists of instances, feasibility rules, and an objective function. A heuristic is a procedure that seeks good solutions without a guarantee of global optimality. A metaheuristic is a higher-level search strategy, such as local search, evolutionary search, or hybrid search, that guides lower-level moves or constructive choices. A hyper-heuristic operates one level higher again by selecting, generating, or adapting heuristics. The proposed LLM-guided loop belongs to this last category because the searched object is executable heuristic code.

2.2 Heuristics, Hyper-Heuristics, and Algorithm Selection

The practical difficulty of many combinatorial optimization problems has long made heuristic design an essential part of computer science and operations research. Rice's classic formulation of the algorithm selection problem cast the issue in its most general form: given a space of instances, a portfolio of candidate algorithms, and a performance measure, the task is to learn or construct a mapping from instances to algorithms that performs well in expectation. In modern notation, if $\mathcal{I}$ is an instance space, $\mathcal{A}$ is a portfolio of algorithms, and $\ell(a, x)$ is a local lower-is-better loss for applying algorithm a to instance x, then a selector σ maps instances to algorithms.

Term	Meaning in this thesis
Feasibility	Satisfaction of task constraints, such as serving every CVRP customer exactly once and respecting route capacity.
Held-out evaluation	Evaluation on instances not used for prompt construction, training feedback, or incumbent selection.
Replay	Reuse of archived failures, summaries, or previous examples in later prompts.
Validator pressure	The strictness imposed by syntax checks, sandbox execution, feasibility validation, scoring, and time limits.
Epoch	One proposal–validation–archive-update cycle in the code-evolution loop.
Paired offset	A shared run index or seed alignment used to compare conditions under matched stochastic or benchmark variation.
Aggregate report	A phase-level artifact that pools repeated runs and reports means, paired deltas, confidence intervals, and diagnostic checks.
Bootstrap comparison	Nonparametric resampling of paired run-level deltas to estimate 95% uncertainty intervals.
Archive diversity	A diagnostic measure of the spread of retained artifacts or archive descriptors within a phase.

Table 2.1: Core terms used consistently in the methodology and evaluation chapters.

With $\mathcal{D}$ denoting the instance distribution, the ideal selector is

$$\sigma^\star \in \arg\min_{\sigma:\mathcal{I}\to\mathcal{A}} \mathbb{E}_{x\sim\mathcal{D}}[\ell(\sigma(x), x)]. \tag{2.1}$$

Equation (2.1) is a compact restatement of Rice's formulation [26]. It remains foundational because it separates the underlying optimization problem from the meta-level question of which procedure should be applied to which instance. The symbols in this background equation are local to algorithm selection. Chapter 3 defines the dissertation-specific aggregate $J_{\mathcal{D}}(h)$ for executable heuristic programs.

Hyper-heuristics emerged as one prominent response to this meta-level challenge. They search over heuristics or heuristic components, with the goal of automating heuristic selection, sequencing, or generation [8, 13]. The shift from solution space to heuristic space is especially relevant for the present dissertation because the proposed LLM-guided loop also operates primarily at the level of programs that produce solutions. The empirical object is therefore the heuristic procedure that generates a solution for each instance. Recent surveys emphasize that hyper-heuristics now span selection, generation, portfolio, and configuration settings, and that modern work increasingly integrates machine learning into all four categories [13]. The present dissertation belongs to this tradition, with the difference that the search operator is a language model capable of proposing new executable code, where earlier systems often use classical mutation, grammar, or rule-combination mechanisms.

Algorithm selection has likewise evolved toward both feature-based and feature-free approaches. Traditional methods rely on hand-designed instance descriptors, whereas newer methods attempt to learn directly from raw instance representations [2]. This contrast frames a key design choice in the adaptive heuristic portfolio experiments: should a controller rely on explicit descriptors crafted from domain knowledge, or can a more generic representation learn the selector implicitly?

2.3 Formal Benchmark Problem Definitions

The empirical phases of this dissertation are built around standard benchmark optimization families. A concise formal account is useful before turning to learning-based methods.

2.3.1 Symmetric Traveling Salesperson Problem

In the symmetric traveling salesperson problem (TSP), one is given a complete undirected weighted graph $G = (V, E, w)$, where V is the set of cities, E is the set of undirected edges, and $w : E \to \mathbb{R}_{\geq 0}$ assigns a non-negative travel cost to each edge. The symmetric case has $w_{ij} = w_{ji}$ for every pair of cities and seeks a minimum-cost Hamiltonian cycle. Using binary edge-selection variables x_{ij}, where $x_{ij} = 1$ means that edge (i, j) is included in the tour and $x_{ij} = 0$ otherwise, the standard subtour-elimination formulation used here follows the Dantzig–Fulkerson–Johnson model, written in symmetric-edge notation for the TSPLIB benchmark setting:

$$\min_{x} \quad \sum_{(i,j)\in E} w_{ij} x_{ij} \tag{2.2}$$

$$\text{s.t.} \quad \sum_{j\neq i} x_{ij} = 2, \qquad \forall i \in V, \tag{2.3}$$

$$\sum_{i,j\in S,\ i<j} x_{ij} \leq |S| - 1, \qquad \forall S \subset V,\ 2 \leq |S| < |V|, \tag{2.4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in E. \tag{2.5}$$

Equations (2.2)–(2.5) are the Dantzig–Fulkerson–Johnson subtour-elimination model written in symmetric benchmark notation [12, 25]. Here, S denotes a proper subset of cities. Constraint (2.3) enforces degree two at every city, while (2.4) removes disconnected subtours. The TSP is NP-hard, which explains why heuristic methods remain central despite decades of exact and approximate progress.

2.3.2 Asymmetric Traveling Salesperson Problem

The asymmetric TSP (ATSP) replaces the undirected graph by a complete directed graph $G = (V, A, c)$. The city set V has the same role as in the symmetric case,

while directed arcs A replace undirected edges E and directed costs $c : A \to \mathbb{R}_{\geq 0}$ replace symmetric weights w. Because c_{ij} and c_{ji} may differ, the goal is to find a minimum-cost Hamiltonian tour with one incoming and one outgoing arc per city. Many constructive ideas transfer from TSP to ATSP, yet the asymmetry changes the local-search geometry and often weakens heuristics that rely on symmetric exchange structure. The missing exchange structure makes ATSP a useful transfer family for testing whether a learned or evolved heuristic reflects a reusable search principle across related routing problems.

2.3.3 Capacitated Vehicle Routing Problem

In the capacitated vehicle routing problem (CVRP), a depot node 0, customer set $V = \{1, \ldots, n\}$, customer demands $q_i > 0$ for each customer i, and vehicle capacity Q are given. The task is to partition customers into depot-rooted routes whose total demand does not exceed Q and whose combined travel cost is minimal. The compact route-based view used in Equation (2.6)–(2.8) follows that standard benchmark objective and capacity structure:

$$\min_{\mathcal{R}} \quad \sum_{R \in \mathcal{R}} c(R) \tag{2.6}$$

$$\text{s.t.} \quad \bigcup_{R \in \mathcal{R}} R = V, \qquad R \cap R' = \varnothing \text{ for } R \neq R', \tag{2.7}$$

$$\sum_{i \in R} q_i \leq Q, \qquad \forall R \in \mathcal{R}. \tag{2.8}$$

Equations (2.6)–(2.8) state the route-based CVRP objective, customer-partition, and capacity constraints used for the benchmark setting considered here [10, 32]. Here, $\mathcal{R}$ is the set of vehicle routes, R is one route in that set, and $c(R)$ is the travel cost of route R. Unlike TSP, CVRP combines sequencing and packing structure. That added constraint pressure is relevant here because a program that looks promising under tour construction alone may fail once feasibility management, route merging, and penalty handling become part of the evaluator.

2.4 Machine Learning for Combinatorial Optimization

Machine learning for combinatorial optimization has developed into a substantial field in its own right. Bengio et al. provide a useful organizing view: optimization problems can be treated as data points, and learning can be used to replace costly or poorly defined decisions inside larger solving procedures [5]. The framing helps avoid the misleading idea that learning must replace classical optimization wholesale. Instead, machine learning can support branching, ranking, repair, construction, selection, or parameterization within a broader pipeline.

A complementary research stream studies direct learning-based solvers. Neural combinatorial optimization with reinforcement learning showed early promise on Euclidean TSP by training neural policies to construct tours directly from coordinates [4]. Khalil et al. extended this direction by learning greedy algorithms over graph-structured state representations, showing that learned policies can act as reusable meta-algorithms across instances [11]. Subsequent work extended learning-based optimization to more constrained settings and more realistic objectives. The broader lesson remained the same: learned solvers can be effective, yet strong classical heuristics often remain difficult to improve across wide instance distributions [5]. This dissertation responds to that observation by studying iterative improvement of heuristic code under benchmark feedback.

The routing literature also provides the baselines and datasets that make such claims meaningful. TSPLIB remains the canonical reference library for traveling salesperson problems [25]. For symmetric TSP, the Lin–Kernighan family and its refinements, including Helsgaun's implementation, remain among the strongest and most influential heuristic baselines [15, 20]. For CVRP, Clarke and Wright's savings heuristic remains a foundational constructive baseline [10], while modern state-of-the-art practice includes far stronger hybrid metaheuristics such as Vidal's hybrid genetic search [33]. The modern Uchoa instance family likewise provides a more challenging and statistically informative benchmark regime than many older CVRP sets [32]. Together, these references define the level of baseline strength against which LLM-evolved heuristics must be judged. A language-model system that improves over naive constructive baselines still needs calibration against the best established heuristics in the literature.

2.5 Large Language Models for Code Generation

The empirical framework studied in this dissertation depends directly on the rapid improvement of code-capable language models. Codex demonstrated that large language models trained on code could solve a meaningful fraction of functional programming tasks and that repeated sampling substantially boosts success rates on harder problems [9]. Austin et al. likewise showed that program synthesis performance scales strongly with model size and that natural-language feedback can materially improve generated programs [3]. Together, these findings indicate that outer-loop search and repair are integral components of strong performance on demanding code-generation tasks.

As the field matured, surveys began to distinguish between simple code completion, benchmarked program synthesis, repair, debugging, and agentic multi-step workflows. Early overviews of NL2Code models emphasized training data, scaling, and benchmark design [35], while more recent surveys document a much larger and more heterogeneous landscape of code models, evaluation protocols, and practical risks [18]. At the same time, stronger benchmarks such as AlphaCode-style

competitive programming and LiveCodeBench reveal that the difference between solving simple textbook-style tasks and solving harder, more recent coding problems remains substantial [17, 19]. This gap reinforces the need to evaluate code generation under task-specific execution and held-out tests.

2.6 Evaluator-Guided Search and Algorithm Discovery

One-shot code generation is only one end of the design spectrum. A more relevant line of work for this dissertation combines generative models with evaluators that score candidate programs and feed the results back into an outer search loop. In abstract form, evaluator-guided program-search systems such as FunSearch can be described as searching a program space $\mathcal{H}$ for

$$h^\star \in \arg\min_{h \in \mathcal{H}} J_{\mathcal{D}}(h), \tag{2.9}$$

Equation (2.9) is used here as an abstract form of evaluator-guided program search, consistent with the role of a frozen LLM and external evaluator in FunSearch [27]. Here, $J_{\mathcal{D}}(h)$ denotes a panel-level performance aggregate over benchmark panel $\mathcal{D}$. Chapter 3 gives the dissertation-specific convention. The framing aligns the generative model with verification instead of free-form textual plausibility.

FunSearch is a central example. It pairs a frozen LLM with a systematic evaluator and a diverse program archive in order to discover high-scoring programs for problems that are hard to solve but easy to evaluate [27]. Its formulation is relevant here because the LLM is used as a variation operator inside an evolutionary process, while the evaluator guards against incorrect or hallucinated proposals.

AlphaDev demonstrates a related idea from a reinforcement-learning perspective by formulating algorithm discovery for low-level sorting routines as a single-player game and optimizing directly for correctness and latency at the assembly level [22]. AlphaCode, while not an evolutionary system in the same sense, also depends heavily on large-scale sampling, behaviour-based filtering, and selection among many candidate programs [19]. The systems reviewed in this section show that modern code-generation systems often rely on the interaction between generative diversity and external verification.

The most direct predecessor to the present dissertation in combinatorial optimization is the study of LLMs as evolutionary optimizers by Liu et al. [21]. Their system treats the LLM as a crossover-and-mutation operator within a population-based optimizer and shows promising TSP results on very small instances. The work is conceptually relevant. Its scale also exposes a gap: there remains a need for research that studies executable whole-program heuristic evolution on standard benchmark families, with stronger baselines, repeated trials, transfer evaluation, and mechanism controls. A recent systematic review of LLMs in combinatorial optimization confirms both the rapid expansion of the area and the diversity of roles

LLMs now play in optimization systems [28]. The review also shows that careful cross-family empirical evaluation remains comparatively rare.

2.7 Curriculum, Replay, and Diversity Mechanisms

The replay-aware components of this dissertation draw most directly from replay mechanisms, diversity-preserving search, and iterative adversarial practice. Curriculum learning provides related background for the early game-policy phases, where the exposure distribution across opponents and held-out panels is part of the experimental design. Bengio et al. formalized the idea that ordering examples from easier to harder can improve learning and generalization [6]. Later surveys and automated curriculum methods extended the idea to a wide range of machine learning and reinforcement learning settings [14, 31]. In adversarial or sequential domains, curricula matter because the learner's exposure distribution is part of the effective training algorithm.

Replay mechanisms provide the closest motivation. In reinforcement learning, prioritized experience replay showed that non-uniform reuse of past transitions can improve learning efficiency relative to uniform replay [29]. In that setting, transitions with larger priority scores are sampled more often, with a prioritization exponent controlling how strongly replay departs from uniform sampling. Later work generalized this idea to sequence-level replay and more structured replay decisions [7]. In the present dissertation, replay means the reuse of instances, failure cases, and archive summaries as a memory mechanism for future program proposals. The analogy motivates direct evaluation of replay in program-level heuristic evolution.

Diversity-based search provides a second mechanism-level motivation. Quality-diversity methods such as MAP-Elites and novelty-search-with-local-competition retain a collection of high-performing, behaviourally distinct artifacts while optimizing the task objective [23, 24]. This family of ideas is relevant because a code-evolution loop can easily collapse to narrow local search unless the archive preserves multiple useful regions of the behavioural space. The present dissertation draws on QD-style intuitions when it tracks novelty, maintains archives, and evaluates whether diversity corresponds to genuine adaptation.

Finally, self-play and adversarial practice provide a motivating example for why curriculum and iterative pressure can induce capability growth at all. AlphaGo Zero showed that high-level performance can emerge from repeated self-play without human demonstrations [30]. Its scale and claims differ substantially from those considered here. The underlying intuition remains relevant: iterative interaction can sometimes produce behaviour that is hard to obtain from static supervision alone.

2.8 Methodological Discipline and the Need for Conservative Claims

Another part of the relevant literature concerns how claims are validated. Henderson et al. document the reproducibility problems that arise when stochastic learning systems are evaluated with too few seeds, weak statistical reporting, or poorly matched baselines. Agarwal et al. extend this point by arguing for robust statistical summaries and uncertainty-aware comparisons in deep RL benchmarking. Although these works target reinforcement learning, rather than code evolution, the methodological lesson transfers directly. When a generative system can produce very different candidate programs across runs, best-case anecdotes are not enough.

Following that methodological literature, the dissertation uses paired offsets, aggregate reports, bootstrap comparisons, held-out panels, and mechanism-specific controls. A simple paired comparison between two conditions A and B can be written as

$$\hat{\Delta}_{A,B} = \frac{1}{n} \sum_{i=1}^{n} \left(m_i^{(A)} - m_i^{(B)} \right), \tag{2.10}$$

Equation (2.10) expresses the paired-comparison reporting convention used in the dissertation, following reproducibility guidance for stochastic learning evaluations [1, 16]. Here, n is the number of paired runs or offsets, and $m_i^{(A)}$ and $m_i^{(B)}$ are the matched measurements for conditions A and B on pair i. Uncertainty is estimated by resampling or other repeated-evaluation procedures. A negative result obtained under proper controls can be more informative than a seemingly positive result that confounds memory mechanisms with extra candidate budget.

2.9 Synthesis and Research Gap

The main research traditions from which this dissertation draws are summarized in Table 2.2. The table is used to locate the dissertation's contribution, not to claim that any one literature already answers the empirical question studied here.

The literature reviewed above supports five observations that matter directly for this thesis. Classical optimization already provides strong benchmark libraries and strong heuristic baselines [10, 15, 20, 25, 32, 33]. Hyper-heuristics and algorithm selection offer established ways to reason about search in both heuristic space and solution space [8, 13, 26]. Modern code LLMs are strong enough to generate executable programs, especially when combined with repeated sampling, testing, and repair [3, 9, 17, 19]. Evaluator-guided algorithm discovery with learned code generators is now credible. Existing success stories often depend on specialized evaluators, narrow scopes, or problem formulations that differ from full-program benchmark optimization [21, 22, 27]. Finally, curriculum, replay, and diversity

Tradition	Search object	Main strength	Primary limit	Representative refs.
Classical heuristics and metaheuristics	solutions or route moves	strong benchmarked performance	expensive domain expertise	[10, 15, 20, 33]
Hyper-heuristics and algorithm selection	heuristic choices or generators	explicit heuristic-level reasoning	relies on crafted operators or features	[8, 13, 26]
Learning-based combinatorial optimization	policies, rankings, or solver components	learns recurring structure from data	difficult to improve over strong classical solvers broadly	[4, 5, 11]
Evaluator-guided program search with LLMs	executable programs	open-ended code generation plus verification	variable search, so mechanism claims need strong controls	[21, 22, 27]

Table 2.2: Positioning of the present dissertation relative to adjacent literatures. The thesis sits closest to evaluator-guided program search and also depends on benchmarked routing, hyper-heuristic reasoning, and conservative empirical methodology.

mechanisms are theoretically motivated. Their value in LLM-guided code evolution remains an empirical question for controlled measurement [6, 24, 29].

The dissertation addresses a gap at the intersection of these points. There is still no clear empirical account of how LLM-guided code evolution behaves across multiple benchmark families when evaluated with replicated runs, strong classical baselines, replay alternatives, and strict validator-based transfer tests. The study therefore centers on a conservative cross-family analysis of executable heuristics.

2.10 Chapter Summary

The literature review established the technical context for the dissertation in four steps. It positioned the work within algorithm selection and hyper-heuristics, where the central object of search is a heuristic procedure. It reviewed the benchmark optimization families that make the empirical claims meaningful, including the standard TSP, ATSP, and CVRP formulations and their mature baseline literatures, and situated the project within modern code-generation and evaluator-guided program-search research, where LLMs are most credible when paired with strong external verification. Finally, it showed why replay, curriculum, and diversity mechanisms must be evaluated under careful experimental discipline and not inferred from isolated positive cases. The next chapter turns from that context to the experimental design used to study the gap identified here.

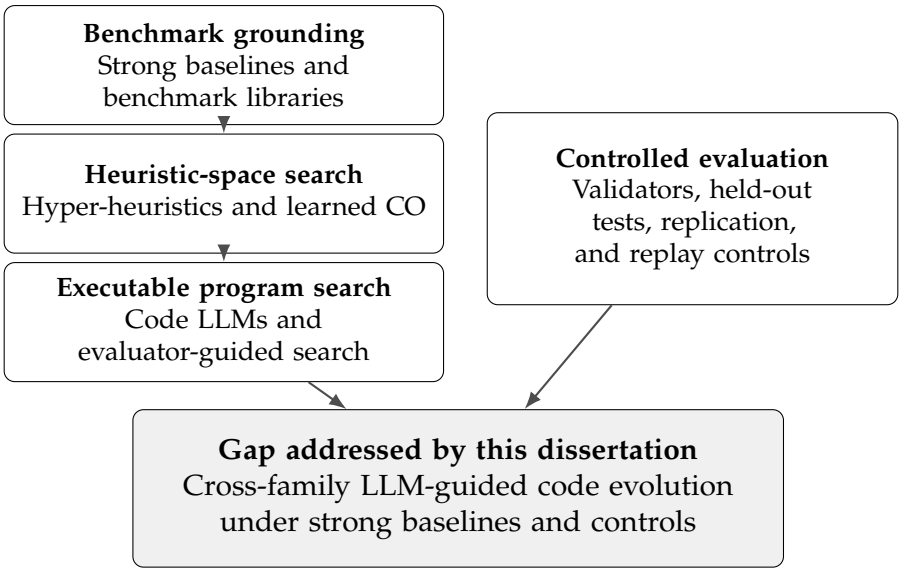

Figure 2.1: Conceptual synthesis of the research threads integrated by the dissertation. The gap is a missing cross-family account that combines benchmark grounding, heuristic-space search, executable-program generation, and strict validator-based evaluation in one empirical framework.

Chapter 3

Conceptual Framework and Methodology

Chapter 3 defines the conceptual framework and experimental design used across the completed phases. The chapter provides the common notation for benchmark families, heuristic programs, evaluation panels, replay, paired comparisons, normalized effect sizes, and artifact availability. The framework is conservative by design: each phase relies on deterministic validators, held-out evaluation panels, repeated runs, and phase-specific technical validity checks before any substantive interpretation is accepted.

3.1 Formal Problem Setting

The study treats heuristic search as a problem of discovering executable programs that map instances to feasible solutions. Let $\mathcal{X}$ denote the concrete benchmark instance set used by a phase, let $x \in \mathcal{X}$ denote one benchmark instance, let $\Omega(x)$ denote the feasible solution set for that instance, and let $f(s; x)$ denote the cost of solution s on instance x. For the routing families, lower cost is better.

Definition 3.1. *A benchmark optimization family is a triple $\mathcal{F} = (\mathcal{X}, \Omega, f)$ consisting of a benchmark instance set $\mathcal{X}$, a feasibility map Ω, and an objective function f.*

Definition 3.2. *A deterministic heuristic program is executable code that, for each benchmark instance $x \in \mathcal{X}$, returns either a feasible solution $h(x) \in \Omega(x)$ or a detectable failure symbol $\perp$ under the fixed interface and resource limits of a phase.*

Definition 3.3. *An evaluation panel is a named subset $\mathcal{D} \subseteq \mathcal{X}$ reserved for a specific evaluation role. A training panel, denoted $\mathcal{D}_{\text{train}}$, is used for candidate feedback and incumbent selection during a run. A held-out or test panel, denoted $\mathcal{D}_{\text{test}}$, is not used for prompt construction or candidate selection and is kept for final claims.*

The cost function f is phase-specific but has the same role throughout: it maps a feasible solution $s \in \Omega(x)$ for instance x to the scalar quantity minimized by the benchmark. For routing, the instance x supplies the benchmark distance matrix $D(x) = (d_{ij}(x))$. If $s = (R_1, \ldots, R_k)$ is a collection of depot-rooted routes, then the routing cost is

$$f(s; x) = \sum_{r=1}^{k} c(R_r; x),$$

where, for a route $R = (v_1, \ldots, v_m)$ whose ordered customer sequence is $v_1, \ldots, v_m$ and whose depot is 0, the single-route cost is

$$c(R; x) = d_{0v_1}(x) + \sum_{j=1}^{m-1} d_{v_j v_{j+1}}(x) + d_{v_m 0}(x).$$

This definition is the cost used by the project validators for TSP-style tours and CVRP route sets.

When a reference cost $f^\star(x)$ is available, the instance-level routing score is the relative gap for a feasible returned solution. In this expression, $h(x)$ is the solution returned by program h on instance x:

$$g(h, x) = \frac{f(h(x); x) - f^\star(x)}{f^\star(x)}. \tag{3.1}$$

This gap convention follows standard benchmark reporting practice for TSP and CVRP libraries [25, 32].

For an evaluation panel $\mathcal{D}$ (Definition 3.3), the main lower-is-better aggregate used in the routing chapters is

$$J_{\mathcal{D}}(h) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} g(h, x). \tag{3.2}$$

Thus $J_{\mathcal{D}}(h)$ measures the average instance-level score of program h on the named panel. For routing gaps and penalized gaps, $J_{\mathcal{D}}(h) \geq 0$ when the reference value is a best-known or optimal cost, and smaller values indicate solutions closer to the reference. A value of 0 means that the program matches the reference cost on every instance in the panel. In CVRP phases, infeasible or non-executable outputs replace the relative gap with a phase-defined penalized gap, so that a solver cannot appear strong by omitting customers, violating capacity, or failing to return a route list. For higher-is-better game endpoints, the sign convention is reversed during standardized comparison so that larger values still indicate better task performance.

The LLM-guided code-evolution loop is the mechanism studied in this thesis. It is defined by a prompt state p_t, an archive state $\mathcal{A}_t$, a task specification $\mathcal{T}$, and an

evaluator E:

$$h_t \sim q_\phi(\cdot \mid p_t), \tag{3.3}$$
$$r_t = E(h_t; \mathcal{D}_{\text{train}}), \tag{3.4}$$
$$\mathcal{A}_{t+1} = U(\mathcal{A}_t, h_t, r_t), \tag{3.5}$$
$$p_{t+1} = C(\mathcal{T}, \mathcal{A}_{t+1}, r_t). \tag{3.6}$$

Here, q_ϕ is the model-conditioned program generator, U is the archive-update rule, and C is the prompt-construction function. Replay changes the information exposed by $\mathcal{A}_{t+1}$ and C, which makes it an experimentally testable mechanism. Equation (3.3) denotes code generation, Equation (3.4) denotes execution and scoring on the training panel, Equation (3.5) denotes the archive update after validation, and Equation (3.6) denotes construction of the next prompt state. These equations formalize the project loop by combining components motivated by the evaluator-guided program-search and replay literatures reviewed in Chapter 2 [27, 29].

For a lower-is-better endpoint, a paired delta between two experimental conditions A and B is reported as

$$\Delta(A, B) = J_{\mathcal{D}_{\text{test}}}(A) - J_{\mathcal{D}_{\text{test}}}(B), \tag{3.7}$$

where $J_{\mathcal{D}_{\text{test}}}(A)$ and $J_{\mathcal{D}_{\text{test}}}(B)$ denote the condition-level test-panel aggregates computed under the same paired design, and negative values favor condition A. The normalized relative improvement (NRI) used in the evaluation tables is

$$\text{NRI}(A, B) = \frac{J_{\mathcal{D}_{\text{test}}}(B) - J_{\mathcal{D}_{\text{test}}}(A)}{|J_{\mathcal{D}_{\text{test}}}(B)|}, \tag{3.8}$$

for comparisons where the comparator aggregate is nonzero. For higher-is-better endpoints, the numerator is reversed. NRI is used only as a scale-normalized effect summary within a comparison, while game score margins, TSP gaps, and CVRP penalized gaps remain distinct endpoint types. The paired-delta and bootstrap reporting conventions follow the reproducibility guidance discussed by Henderson et al. and Agarwal et al. for stochastic learning experiments [1, 16]. The equations here state the specific convention used for this dissertation's archived runs.

3.2 Research Design Overview

The study reports completed empirical campaigns as a staged research sequence. The sequence was chosen so that the search object, benchmark structure, and validator pressure become progressively more demanding. Early phases test whether iterative code evolution can alter behavior at all. Later phases ask whether the same loop can preserve useful adaptations on standard optimization benchmarks with stronger

Phase	Search object	Replicates	Conditions	Primary endpoint
1–4 (simple games)	full deterministic game-policy code	10 transfer runs plus manual review	3 transfer environments and 3 audited curriculum recipes	held-out win rate and score margin
5 (routing replay transfer)	constrained routing heuristic code	20 paired offsets per family	4 replay conditions across TSP, ATSP, and CVRP	held-out benchmark optimality gap
6 (replay mechanism study)	constrained TSP heuristic code	20 paired offsets	7 replay variants	held-out TSPLIB optimality gap
7 (operator discovery)	modular TSP operator code	20 paired offsets	7 conditions	held-out TSPLIB gap plus transplant/ablation validation
8 (adaptive portfolio)	TSP controller over a frozen heuristic portfolio	20 paired offsets	8 conditions	held-out TSPLIB gap and selector regret
9 (real-world solver evolution)	bounded CVRP solver code	20 paired offsets	4 conditions	held-out penalized gap with feasibility pressure
9 closeout (budget control)	bounded CVRP solver code under matched budgets	20 paired offsets	6 conditions	held-out penalized gap with explicit learned controls
12 (SOTA calibration)	external CVRP solver	5 held-out instances, 5 seeds each	PyVRP HGS-style solver	validated held-out CVRP gap
Model factorial	task–model–technique cells	750 rows	3 task families, 3 model tiers, 5 techniques	standardized task performance and budget-control comparisons

Table 3.1: Completed empirical phases and calibration analyses retained in the final dissertation.

baselines and clearer notions of feasibility. Table 3.1 summarizes the phase sequence used in the final dissertation.

The archival phase record is finer-grained than the bundled presentation used in the dissertation. In the project archive, phases 1, 2, 2b, 3, and 4 are preserved as distinct simple-game campaigns. The final manuscript groups them into one simple-game evidence block because they share the same environment family and lead to one retained transfer-and-novelty conclusion. The phase-9 closeout is retained separately as a scoped mechanism study on the same bounded CVRP regime under matched learned controls. The HGS-style calibration is retained as an external baseline analysis outside the evolved-code conditions. The crossed model-strength factorial is retained as a final control analysis because it directly tests whether replay effects survive budget-matched comparisons across three model tiers.

Role or analysis	Model identifiers recorded in the archived configurations
Main generator	`gpt-5.4-nano` for the retained simple-game transfer, routing replay, TSP mechanism, operator-discovery, adaptive-portfolio, CVRP solver, and phase-9 budget-control studies
Support calls	`gpt-4.1-mini` where a phase configuration used a separate repair, judge, or auxiliary model
Model-factorial ladder	`gpt-4.1-nano`, `gpt-5-nano`, and `gpt-5.4-mini`
Local and external controls	not OpenAI API models: local baselines such as `nearest_resource` and `opponent_shadow`, classical routing heuristics, and the PyVRP HGS-style solver

Table 3.2: OpenAI model identifiers and non-LLM controls used in the retained experimental record.

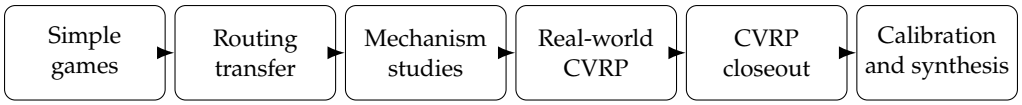

Figure 3.1: Progression of the completed experimental program, from low-cost game adaptation to routing, mechanism studies, higher-level search objects, bounded real-world CVRP, matched-budget closeout, and final calibration/control analyses. Portfolio control is included in the mechanism-study stage.

3.3 Common Code-Evolution Loop

All phases instantiate the code-evolution loop defined in Section 3.1, with shared constraints across families. At each epoch, the model receives a bounded interface, the current incumbent or scaffold, and structured feedback from prior evaluations. It then proposes a deterministic Python artifact: a game policy, routing heuristic, modular operator, controller, or whole CVRP solver. That proposal is validated, executed on the training panel, and either rejected or admitted into the evolving archive.

Across all families, generated code is deterministic and auditable: imports, hidden file or network access, and arbitrary external solver calls are disallowed. Task-specific validators separate syntactic success from scientific usefulness, so a candidate must execute and return a structurally valid output before it can be considered for acceptance. In this dissertation, the inner loop means the repeated proposal–validation–archive-update cycle that uses training probes, replay probes, and archive summaries to guide the next generated artifact. Held-out evidence remains outside that loop: it is frozen until the end of a run and is used only for final claims. Fallback or errored generations are logged diagnostically and are not

silently counted as successful improvements.

These constraints narrow the interpretation of success. A generated program is useful only when it survives validation, remains executable across repeated evaluations, and improves held-out performance under the benchmark-specific metric.

3.4 Family-Specific Search Objects and Benchmarks

The common code-evolution loop is implemented differently across families because the scientific question changes with the object of search. In the simple-game phases, the search object is full policy code for a two-player environment. The environments emphasize transfer and behavioral change. Held-out win rate and score margin are the relevant outcomes.

In the routing replay phases, the search object is a constrained heuristic scaffold. For symmetric TSP and ATSP, the evolved code modifies bounded construction, perturbation, and local-search behavior over benchmark instances derived from TSPLIB [15, 20, 25]. In the initial CVRP extension, the same replay logic is transferred to a constrained multi-route scaffold evaluated on CVRPLIB-style instances [10, 32]. Relative or penalized gap is the primary metric because it standardizes performance across instances of different scales.

Phase 7 changes the search object again, from whole-solvers to modular operators. The aim is to test whether a reusable perturbation, candidate-pruning, or restart component can survive transplant and ablation checks. Phase 8 then switches from operator evolution to algorithm selection: the model evolves a bounded, interpretable controller over a frozen portfolio of classical TSP heuristics, which places the phase directly within the algorithm-selection tradition introduced by Rice [26]. Phase 9 moves to bounded whole-solver evolution on real-world Uchoa X-series CVRPLIB instances [32], where feasibility, repair logic, runtime variability, and route cost all affect interpretation. The phase-9 closeout keeps the same bounded CVRP regime and validator while replacing the original comparison set with matched learned controls designed to distinguish replay-specific value from extra candidate budget or direct one-repair synthesis alone.

3.5 Replication, Aggregation, and Statistical Reporting

The reporting discipline follows the conservative empirical guidance discussed in Section 2.8. The routing, operator, portfolio, and real-world CVRP phases were all run with fixed paired seed offsets, usually twenty offsets shared across compared conditions. The design supports paired deltas of the form in Equation (3.7) and reduces the chance that one condition appears stronger only because it benefited from an easier stochastic draw.

The aggregate reports emphasize condition means on the primary held-out endpoint, paired condition deltas with 95% bootstrap intervals, technical-validity summaries including fallback counts, generation errors, timeout checks, and acceptance counts, and diagnostic metrics such as code novelty, adaptation efficiency, archive hardness, or runtime inflation when those metrics are relevant to the specific phase. The format keeps the primary endpoint, uncertainty estimate, and validity evidence visible together in one report.

The study does not rely on best-run anecdotes. When a phase produces a strong positive result, that result is reported together with the paired uncertainty and with the corresponding validity checks. When a phase produces a null or boundary result, that outcome is reported as a finding.

The labels used in the empirical chapters follow a fixed interpretation rule. A strong positive result requires an improvement on the phase's primary endpoint, paired uncertainty that excludes zero when a paired design is available, and successful technical validity checks. A boundary result indicates a favorable direction with an interval that overlaps zero, a secondary endpoint, or validity evidence that is too narrow for a stronger claim. A null or negative result indicates no credible improvement on the retained endpoint. These labels are interpretive summaries of the reported statistics and audits rather than hidden automated thresholds.

3.6 Qualitative Audit and Validity Safeguards

Scalar metrics do not answer every relevant question. The first additional safeguard is manual novelty adjudication in the simple-game transfer work. The largest code novelty spikes are reviewed and classified as functional adaptation, mixed or local adjustment, or churn/failed adaptation. The audit guards against the common mistake of equating large lexical change with strategically meaningful change.

The second safeguard is a set of phase-specific technical validity audits before interpretation. Examples include checking that all paired runs and all conditions are present, verifying that accepted artifacts did not depend on fallback behavior, and confirming that operator or solver validation pipelines executed as intended. The phase-7 operator campaign, for instance, was deemed ready for analysis only after confirming zero modular generation fallbacks and zero materialization fallbacks in the official run set. Phase 9 was likewise interpreted only after checking full held-out feasibility, zero accepted fallback generations, and the absence of hidden solver-worker timeout contamination in the completed official runs. The phase-9 closeout adds one more safeguard to the same regime: replay is credited only after comparison against matched learned controls that remove the simpler explanations of extra search budget or fresh direct synthesis.

3.7 Scope, Artifact Availability, and Reproducibility

The final manuscript is restricted to completed phases whose evidence is coherent and whose reporting pipeline is already established. The scope choice follows the dissertation's methodological position: the objective is a defensible account of when the code-evolution loop adapts, when it churns, and how those patterns change as validators and baselines become harder.

For computational reproducibility, each reported claim is tied to an official archived run artifact and, where applicable, to a tagged code snapshot in the project version-control record. The accompanying repository preserves the executable code, configuration files, dependency records, benchmark-preparation scripts, phase protocols, aggregate reports, selected generated artifacts, and tagged phase archives used to support the thesis claims. Versioned tags and archived aggregate reports, rather than mutable branch state, are the reference points for the reported evidence.

The public run archive retains experiment-level provenance records from the official runs: prompt text sent to the model, model responses, extracted candidate JSON files, generated policy or solver code, validation records, and fallback/error records. These files connect the reported aggregate tables to the evolved programs that produced them. The public archive excludes credentials, local environment files, private revision notes, local cache files, and large transient worker outputs that are not needed to interpret the reported aggregates.

Chapter 4

Experimental Evaluation and Results

This chapter consolidates the experimental setup, empirical results, and interpretation of the completed campaigns. The organization follows the progression of the research program: early adversarial transfer studies, routing replay studies, higher-level TSP search objects, bounded real-world CVRP solver evolution, and final cross-family control analyses. All reported endpoint values come from archived run artifacts, aggregate reports, or the reproducible external calibration baseline.

4.1 Master Evidence Atlas

Table 4.1 gives the retained empirical record in one place. The table uses the sign convention introduced in Chapter 3: for lower-is-better gap metrics, negative deltas favor the LLM-evolved or replay condition, while positive deltas favor the comparator. The normalized relative improvement (NRI) is interpreted only within a row because the underlying endpoints differ across families. The abbreviation CI denotes a 95% bootstrap confidence interval.

Table 4.1: Master evidence atlas across the retained empirical campaigns. The "effect" column reports the main delta and, when directly comparable, the normalized relative improvement.

Campaign	Family	Search object	Conditions / pairing	Endpoint, comparison, and effect	Interpretation
Simple-game transfer	games	policy code	ten runs with manual novelty adjudication	Held-out win rate and score margin. Novelty spikes are compared with retained behavioral value in Table 4.2.	The loop can alter behavior. Code novelty alone is insufficient evidence of useful adaptation.
Phase 5	TSP	heuristic code	four replay arms, twenty paired offsets	Held-out TSPLIB gap. Random replay mean 0.152211 versus no replay. $\Delta = -0.0311$, CI $[-0.0602, -0.0008]$, NRI $\approx 17.0\%$.	The most statistically separated replay result appears on symmetric TSP and comes from random replay.
Phase 5	ATSP	heuristic code	four replay arms, twenty paired offsets	Held-out ATSP gap. Compressed replay mean 0.187225 versus no replay. $\Delta = -0.0056$, CI $[-0.0191, 0.0071]$, NRI $\approx 2.9\%$.	The signal is weak and not cleanly separated on the primary endpoint.
Phase 5	CVRP	route code	four replay arms, twenty paired offsets	Held-out CVRPLIB gap. No replay is best with mean 0.235779. Replay-family arms are mixed.	The replay mechanism does not transfer as a default rule from TSP to CVRP.
Phase 6	TSP	heuristic code	seven replay variants, twenty paired offsets	Held-out TSPLIB gap. Diversity-aware versus raw failure replay. $\Delta = -0.008637$, CI $[-0.035972, 0.017862]$.	Archive curation is more important than archive size or raw failure accumulation.
Phase 7	TSP	operator code	seven conditions, transplant and ablation checks	Held-out TSPLIB gap plus validation checks. Modular evolution versus heuristic-only baseline. $\Delta = -0.003035$, CI $[-0.010147, 0.000716]$.	Rediscovery signatures recur. Reusable-operator validation fails.
Phase 8	TSP	portfolio controller	eight conditions, twenty paired offsets	Held-out TSPLIB gap and selector regret. Adaptive controller versus static LLM selector. $\Delta = -0.038118$, CI $[-0.06868, -0.00707]$.	Adaptive control improves over a weak selector. Low portfolio headroom prevents improvement over the best fixed heuristic.
Phase 9	CVRP	solver code	four conditions, twenty paired offsets	Held-out penalized gap. Solver evolution versus Clarke–Wright. $\Delta = +0.108465$, CI $[0.071184, 0.144731]$.	Evolved solvers remain feasible and improve over weaker baselines while trailing the strongest fixed heuristic.

Campaign	Family	Search object	Conditions / pairing	Endpoint, comparison, and effect	Interpretation
Phase 9 closeout	CVRP	solver code	six conditions, twenty paired offsets	Held-out penalized gap. Replay evolution versus budget-matched no replay. $\Delta = -2.761632$, CI $[-3.369777, -2.201586]$, NRI $\approx 94.2\%$.	Replay-aware incumbent preservation has lower penalized gap than learned controls mainly by avoiding feasibility collapse.
Phase 12 HGS calibration	CVRP	external solver	PyVRP HGS-style solver, five seeds per held-out instance	Held-out penalized gap. PyVRP best-seed mean 0.006197, about 91.6% lower than Clarke–Wright and 96.4% lower than closeout replay.	Modern specialized VRP search remains the stronger reference point, so no practical solver-dominance claim is supported.
Model factorial	mixed	crossed cells	750 rows, three model tiers, five techniques	Standardized task performance. Replay/failure/compression versus budget-matched no replay. 0 positive bootstrap-supported advantages, pooled model-strength $R^2 = 0.000884$.	Apparent gains over single-shot mostly reflect search budget and task-specific interactions.

4.2 Adversarial Transfer and Early Behavioral Evidence

The simple-game phases serve two purposes in the dissertation. First, they provide a low-cost environment in which iterative code evolution can be observed over many epochs without the computational burden of large benchmark routing suites. Second, they make it possible to inspect the relationship between code novelty and behavioral change before moving to harder optimization domains. These phases establish whether the loop can produce transfer-relevant adaptation before any routing-quality claim is considered.

4.2.1 Transfer Setting

The retained simple-game evidence combines two artifacts: the official transfer suite and the phase-4 novelty adjudication. The transfer suite spans ten replicated runs, three transfer environments, and 3000 total epochs. The environments are pursuit/evasion, resource collection/denial, and territory control.

The suite evaluates whether the evolved agents remain effective when the environment changes and when performance is measured on held-out opponent panels. The earlier curriculum search is not reopened here. The main quantitative outcomes are held-out win rate, the fraction of held-out games won, and held-out score margin, the mean score difference on the same opponent panels.

The novelty adjudication provides the complementary qualitative control. It reviews the strongest novelty spikes from three transfer-era curriculum recipes: rotating-opponents holdout endpoint, rotating + nemesis + novelty + replay, and rotating + replay-aware selection.

Each reviewed spike is conservatively classified as functional adaptation, mixed or local adjustment, or churn/failed adaptation. This classification ties the early transfer claim to actual strategy change rather than to different code text alone.

4.2.2 Manual Novelty Adjudication

The manual calls for the top novelty spikes in each recipe are summarized in Table 4.2. These calls provide qualitative evidence about artifact behavior alongside the held-out transfer metrics reported later in the section.

Three points follow from Table 4.2. First, the baseline rotating-opponent recipe is dominated by churn or failed adaptation among its largest code changes. Second, the heavier replay-aware recipes improve the ratio of useful to useless novelty spikes, suggesting added selection pressure can matter. Third, the stronger recipes cannot support a claim of broad strategic invention. Even when they produce functional improvements, those improvements are uneven across environments and remain mixed with several large but unhelpful code changes.

Recipe	Functional	Mixed/local	Churn/failed
Rotating-opponents holdout endpoint	2/10	2/10	6/10
Rotating + nemesis + novelty + replay	5/10	2/10	3/10
Rotating + replay-aware selection	5/10	2/10	3/10

Table 4.2: Manual adjudication of the strongest novelty spikes in the phase-4 transfer-era recipes, with ten audited spikes per recipe. The heavier replay-aware recipes show more genuine functional adaptations than the simple rotating baseline, while substantial churn remains even in the stronger recipes.

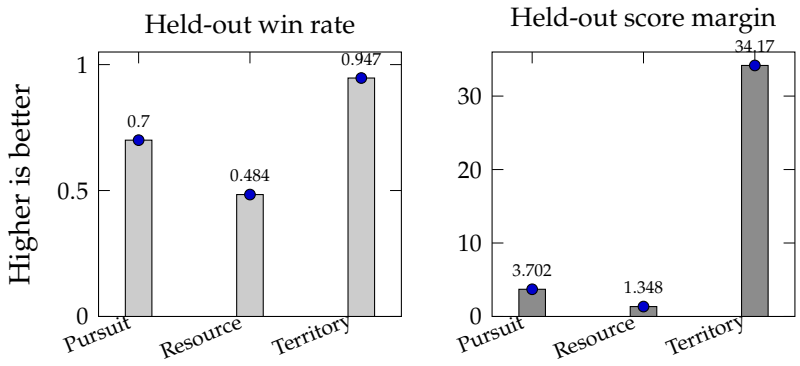

Figure 4.1: Family-level held-out outcomes for the simple-game transfer suite. Territory control shows the strongest positive transfer, pursuit/evasion remains moderately positive, and resource-collection/denial is the weakest environment.

4.2.3 Held-Out Transfer Performance

The official cross-family synthesis reports the family-level held-out performance shown in Figure 4.1. The numbers are aggregated at the environment level because the simple-game role in the dissertation is to establish transfer-capable adaptation across settings.

The three environments differ materially. Territory control shows the most favorable success signal, with held-out win rate 0.9467 and mean score margin 34.17. Pursuit/evasion is weaker but still positive, with win rate 0.70 and margin 3.702. Resource-collection/denial is the most difficult transfer environment, with win rate only 0.484 despite a positive mean margin of 1.348.

The pattern matches the novelty audit. The most convincing functional adaptations were concentrated in the territorial environment, while the other environments showed a larger share of local retuning and churn. The simple-game evidence supports a bounded positive claim: iterative code evolution can produce real transfer-relevant behavioral change. That change is domain-dependent and cannot be inferred from novelty alone. Appendix A.1 and Appendix A.2 provide

representative accepted and rejected artifacts from these early phases.

4.2.4 Interpretation

The simple-game phases support the dissertation in three ways. First, they provide direct evidence for Hypothesis H1 from Section 1.3. The loop does generate executable diversity, and some of that diversity corresponds to useful held-out adaptation.

Second, they establish a methodological caution that remains true in later sections: lexical novelty is an unreliable proxy for functional gain. Without the manual adjudication pass, the early project could easily have overstated the meaning of large code changes.

Third, the game phases motivate the move to routing benchmarks. Once the possibility of adaptation is established in a low-cost environment, the harder question becomes whether the same machinery survives stronger baselines, clearer notions of feasibility, and more standardized benchmark evaluation. The next section addresses that question directly through replicated TSP, ATSP, and CVRP studies.

4.3 Routing Replay and Mechanism Studies

The routing phases move the project from behavioral game environments to benchmark optimization. Routing benchmarks provide stronger baselines, fixed held-out panels, and more standardized notions of progress. The original routing hypothesis was that replay-aware code evolution would improve held-out benchmark gap, and that the improvement would be strongest when replay focused on informative failures. The observed result is narrower.

4.3.1 Phase-5 Routing Transfer Across TSP, ATSP, and CVRP

The first routing campaign compares four replay conditions across three benchmark families: no replay, random replay, raw-gap failure replay, and failure replay with compression pressure. Each family uses twenty paired offsets and reports paired bootstrap intervals on the relevant held-out benchmark endpoint. The resulting evidence is summarized in Table 4.3.

Table 4.3 rules out a simple replay-dominance claim. Replay helps on symmetric TSP, where random replay is the strongest arm. ATSP retains only a weak compression-aware signal, and the primary held-out ATSP endpoint is not cleanly separated. In the replicated phase-5 CVRP suite, no-replay remains the best overall condition.

The aggregate summaries establish the direction of the family-level effects. The offset-level paired deltas show how uneven those effects remain. Figure 4.2 displays the paired held-out deltas for the two phase-5 contrasts that matter most

Family	Best mean condition	Key paired comparison	95% interval	Interpretation
TSP	random replay	held-out TSPLIB delta versus no replay: −0.0311, combined transfer delta: −0.0282	[−0.0602, −0.0008] on TSPLIB and [−0.0520, −0.0029] on combined transfer	statistically separated replay signal, with random replay as the best arm
ATSP	failure replay with compression	held-out ATSP delta versus no replay: −0.0056, combined transfer delta: −0.0063	[−0.0191, 0.0071] on ATSP and [−0.0133, −0.0002] on combined transfer	weakly favorable with interval overlap on the primary endpoint
CVRP	no replay	failure replay versus random replay on held-out CVRPLIB: −0.0076	[−0.0146, −0.0006]	replay is not dominant, and no-replay remains the strongest family-level baseline

Table 4.3: Phase-5 family summaries across the first replicated routing campaign, based on twenty paired offsets per family. Lower deltas are better because all routing endpoints are lower-is-better gap metrics.

to the later interpretation: random replay versus no replay on symmetric TSP, and compression-aware failure replay versus no replay on ATSP.

For the thesis interpretation, the family-level pattern is decisive. A uniform replay effect across TSP, ATSP, and CVRP would support a reusable memory mechanism with broad routing scope. The evidence is narrower: replay pressure interacts with benchmark structure, and the initially favored raw-failure recipe is not the stable best condition even within routing. A representative replay-generated TSP heuristic fragment is shown in Appendix A.3.

4.3.2 Phase-6 TSP Replay Mechanism Study

Phase 6 asks why random replay had lower held-out gap than raw failure replay in the initial TSP study. The follow-up campaign remains within symmetric TSPLIB TSP and expands the mechanism space to curated and diagnostic replay variants, including diversity-aware failure replay, residual-failure replay, and compression applied to the stronger replay arms.

Phase 6 rejects the simple explanation that broader replay coverage alone accounts for the phase-5 TSP outcome. Diversity-aware failure replay has the best mean result, and the archive diagnostics show that hardness tracks final held-out TSPLIB gap more strongly than descriptor diversity does. Table 4.4 gives the most important mechanism comparisons.

The diagnostic side of phase 6 helps resolve the interpretation. Archive diversity has near-zero association with held-out TSPLIB gap (Pearson $r = -0.009478$, Spearman $\rho = -0.041013$), while archive hardness has a materially stronger positive association with worse held-out gap (Pearson $r = 0.39687$, Spearman $\rho = 0.371779$).

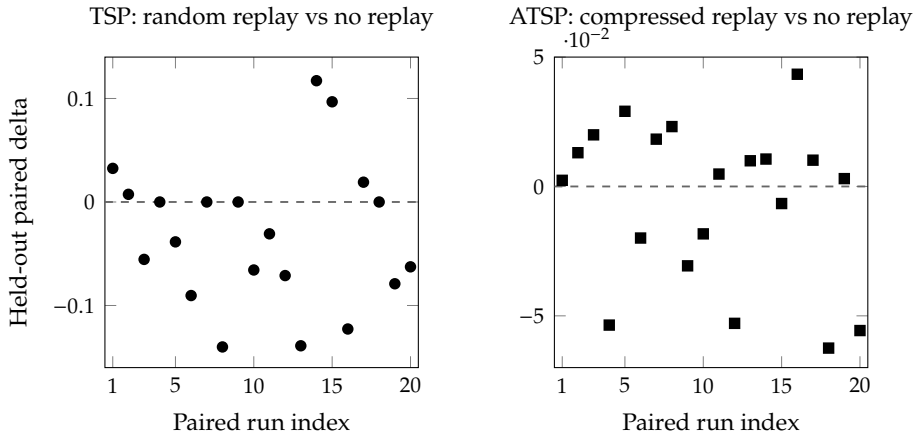

Figure 4.2: Per-run paired held-out deltas for the two phase-5 replay contrasts most relevant to the later thesis interpretation, using twenty matched offsets in each family. Negative values are better because the endpoint is lower-is-better gap. The TSP replay signal is real and uneven across offsets, while the ATSP compression effect remains visibly mixed.

Here, r is the Pearson correlation coefficient and ρ is Spearman's rank correlation coefficient. These correlations weaken a diversity-maximization explanation for the random-replay advantage. Naive failure replay appears to concentrate too heavily on hard cases, while curated replay recovers some of that loss without producing a decisive universal advantage.

4.3.3 Implications of the Routing Phases

Phases 5 and 6 refine the dissertation's replay claim substantially. They show that replay can be useful under conditions shaped by the benchmark family, archive policy, and endpoint being measured.

Replay can help on benchmark routing under specific conditions. Raw failure replay is not a reliable default, and curation is more important than failure accumulation alone. Compression reduces novelty growth without improving the best replay arms in the completed TSP mechanism study. The same replay logic also does not transfer cleanly across TSP, ATSP, and CVRP.

The mixed findings are methodologically useful. They limit the replay claim and motivate the later phases that change the search object instead of adding more replay variants to the same scaffold.

Comparison	Transfer delta	95% interval	Reading
random replay vs. no replay	−0.003123	[−0.027964, 0.021061]	slight mean advantage for random replay with no decisive separation
raw failure replay vs. random replay	+0.001361	[−0.022536, 0.025457]	naive raw-failure replay does not recover the phase-5 TSP advantage
diversity-aware failure replay vs. raw failure replay	−0.008637	[−0.035972, 0.017862]	curation improves the mean arm without decisive paired separation
residual-failure replay vs. raw failure replay	+0.022578	[−0.005643, 0.04867]	residual-failure replay underperforms the naive raw-failure arm
random replay with compression vs. random replay	+0.014874	not favorable	compression lowers novelty but worsens transfer on the random-replay arm
diversity-aware failure replay with compression vs. diversity-aware failure replay	+0.021526	not favorable	compression also hurts the best diversity-aware replay arm

Table 4.4: Key phase-6 TSP replay-mechanism comparisons, based on twenty paired offsets. Positive deltas are worse because the metric is lower-is-better transfer gap.

4.4 Operator Discovery, Portfolio Control, and Real-World CVRP Studies

The later phases use stricter search objects and validation rules. Beyond asking whether replay can help a fixed scaffold, they ask whether the search can discover reusable operators, learn an interpretable controller over a frozen portfolio, synthesize feasible solver logic on real-world CVRP instances, or survive explicit matched-budget controls on that same bounded CVRP regime. These phases separate bounded positive results from claims the evidence cannot support.

4.4.1 Phase 7: Modular Operator Discovery

Phase 7 changes the search object from a whole TSP heuristic to a single modular operator. The official campaign contains twenty paired runs, 800 modular epochs, zero modular generation fallbacks, zero operator materialization fallbacks, and zero ‘missing_build_operator‘ failures. The scientific question is stricter than in the routing replay phases. A candidate operator is not counted as a discovery unless it survives transplant across multiple scaffolds, family-level evaluation, ablation, and Pareto-style practicality checks. Table 4.5 summarizes the main outcome.

Rediscovery and validated discovery are separate claims. Rediscovery means that similar operator families reappear across seeds, so the search is repeatedly returning to the same operator templates. Validated discovery requires more: a reusable operator must show positive transplant evidence, family-level benefit, acceptable runtime behavior, and ablation support for the operator itself. No candidate met all of those criteria in the completed campaign.

Phase 7 is a negative result on the retained endpoint. Its value is methodological: it shows that the validation pipeline rejects brittle or scaffold-specific artifacts

Item	Result
Best overall condition	full-solver evolution, transfer gap 0.117916, TSPLIB gap 0.164668
Best modular condition	modular operator with random replay, transfer gap 0.118833, TSPLIB gap 0.230474
Modular evolution vs heuristic-only baseline	transfer delta −0.001715, 95% CI [−0.003418, 0.000085]. TSPLIB delta −0.003035, 95% CI [−0.010147, 0.000716]
Modular evolution vs full-solver evolution	TSPLIB delta +0.067355, 95% CI [0.040498, 0.094769] in favor of full-solver evolution
Validation outcome	no candidate satisfied the conservative surviving-operator rule
Rediscovery signal	56 distinct rediscovery signatures. The most common family was a double-bridge segment-reversal perturbation

Table 4.5: Phase-7 operator-discovery summary, based on twenty paired offsets. The search generates recurrent operator families, and no operator survives the full validation pipeline.

and prevents an unsupported reusable-operator claim. Appendix A.4 shows a representative rediscovered double-bridge perturbation artifact from this phase.

4.4.2 Phase 8: Adaptive Heuristic Portfolio Control

Phase 8 reframes the problem as algorithm selection. The model evolves a bounded and interpretable controller over a frozen portfolio of known TSP heuristics. The target is more conservative. If phase 7 suggests that reusable low-level innovation is hard to validate, then a higher-level selection problem may still admit useful learning.

The completed campaign contains twenty paired runs, eight compared conditions, judge support in every run, and only three controller-generation errors across the whole replay-aware arm. All three of those errors occurred in rejected epochs, so no accepted final controller depends on fallback behavior. Table 4.6 gives the key phase-8 comparisons.

The central structural result is the first row of interpretation in Table 4.6: the oracle selector ties the best single fixed heuristic on the primary held-out TSPLIB endpoint. In algorithm-selection terms, the frozen portfolio has almost no oracle headroom on that endpoint. If the oracle is not better than the single best fixed heuristic, then no learned controller should be expected to produce a strong positive selector-regret result there.

The structural constraint explains the mixed phase-8 outcome. The adaptive controller learns a useful correction relative to a weak one-shot LLM rule baseline. It still remains behind the strongest fixed or supervised baselines on the primary endpoint. Controller learning is not the only limitation. The frozen portfolio also lacks complementary headroom on the primary endpoint.

Comparison or statistic	Result
Best held-out TSPLIB condition	best single fixed heuristic, mean gap 0.07226
Oracle selector vs best fixed heuristic	held-out TSPLIB delta 0.0, transfer-gap delta −0.004544
Supervised selector vs best fixed heuristic	held-out TSPLIB delta 0.0, transfer-gap delta 0.0
LLM static selector vs supervised selector	held-out TSPLIB delta +0.121255, 95% CI [0.099278, 0.140226]
Adaptive controller vs LLM static selector	held-out TSPLIB delta −0.038118, 95% CI [−0.06868, −0.00707]
Replay-aware adaptive controller vs no-replay adaptive controller	held-out TSPLIB delta +0.010114, 95% CI [−0.020641, 0.041674]
Full-solver evolution vs adaptive controller	held-out TSPLIB delta +0.011853, 95% CI [−0.025032, 0.049041]

Table 4.6: Phase-8 adaptive-portfolio summary, based on twenty paired offsets. Lower deltas are better because the primary endpoint is held-out TSPLIB gap.

Training instances	Held-out instances
X-n101-k25, X-n110-k13, X-n125-k30, X-n134-k13, X-n157-k13, X-n214-k11	X-n176-k26, X-n190-k8, X-n223-k34, X-n247-k50, X-n275-k28

Table 4.7: Phase-9 bounded Uchoa X-series CVRPLIB split, with six training instances and five held-out instances. The train/held-out separation is fixed before the campaign and is not reopened during solver evolution.

4.4.3 Phase 9: Real-World CVRP Solver Evolution

Phase 9 is the most realistic completed campaign in the dissertation. The search object is now a bounded whole solver for Uchoa X-series CVRPLIB instances [32]. The solver must return a full route set, satisfy customer-coverage and vehicle-capacity constraints, and tolerate stronger validator pressure than the earlier routing scaffolds. The benchmark split used in the official bounded suite is listed in Table 4.7.

The official phase-9 campaign contains all four conditions in every paired run, judge support remains enabled throughout, and the accepted solvers do not rely on fallback code or hidden timeout failures. Across the whole campaign, the solver-evolution arm records 39 accepted epochs out of 160 total epochs, 0 generation fallbacks, 0 accepted errored generations, and full held-out feasibility in every official run. Table 4.8 summarizes the main held-out outcomes.

The paired comparisons quantify the difference more directly. Solver evolution has a held-out penalized gap that is lower than nearest-neighbor by 0.091204 penalized-gap units. The paired 95% CI is $[-0.128136, -0.055417]$. It also improves on regret-insertion plus local search by 0.284160 penalized-gap units, with paired 95% CI $[-0.321769, -0.247706]$. Against Clarke–Wright, the sign reverses: solver evolution is higher by 0.108465 penalized-gap units, with paired 95% CI $[0.071184, 0.144731]$.

Condition	Held-out feasibility	Mean penalized gap	Reading
Clarke–Wright savings	1.0	0.073766	strongest fixed baseline
Solver evolution	1.0	0.182231	bounded positive result with higher gap than Clarke–Wright
Nearest-neighbor constructive	1.0	0.273435	weaker fixed baseline
Regret insertion plus local search	1.0	0.466391	weakest baseline on the primary endpoint

Table 4.8: Phase-9 held-out results on bounded real-world CVRP, based on twenty paired runs over the five held-out Uchoa X-series instances. Lower penalized gap is better.

Two additional properties affect interpretation. First, the solver-evolution arm remains fully feasible on both train and held-out panels throughout the completed campaign. That means the loop preserves feasible executable solver updates under validator pressure. Second, the resulting solver set is not one repeated artifact. There are sixteen distinct final solver fingerprints across the twenty official runs, although some no-acceptance runs remain at the same starting incumbent.

Runtime variability limits the result. The evolved solvers have mean held-out runtime about 821.7 ms, median runtime about 255.2 ms, and a heavy upper tail above seven seconds in the worst run-level case. Phase 9 does not support a claim of practical dominance. It is a bounded real-world result: given only the specification, validator, scorer, and training instances, the loop can generate feasible held-out solver logic that improves over weaker baselines without surpassing the strongest fixed heuristic in the benchmark regime.

The runtime profile also clarifies the quality trade-off. The evolved solvers improve the primary held-out gap relative to the weaker constructive baselines while preserving feasibility. Their wall-clock cost is materially less predictable than that of the simpler fixed heuristics. The phase-9 result is evidence of bounded solution-quality adaptation under validator pressure.

Figure 4.3 makes those boundary conditions explicit at the run level. The train-versus-held-out scatter shows that the stronger phase-9 runs stay strong off the training panel without a complete out-of-sample collapse. The held-out relationship remains variable. The runtime panel shows a heavy-tailed cost profile, and the paired delta panels show why the phase-9 result remains bounded: solver evolution usually has lower gap than the weaker nearest-neighbor baseline and only rarely has lower gap than Clarke–Wright.

Figure 4.4 adds two diagnostics that are not visible in the final held-out mean alone. First, accepted updates are sparse and spread across the entire eight-epoch budget and are not concentrated at the start of the run. Second, the instance-level contrast is uniform in sign: averaged across the twenty official runs, solver evolution has lower gap than nearest-neighbor on all five held-out Uchoa X-series instances and higher gap than Clarke–Wright on all five.

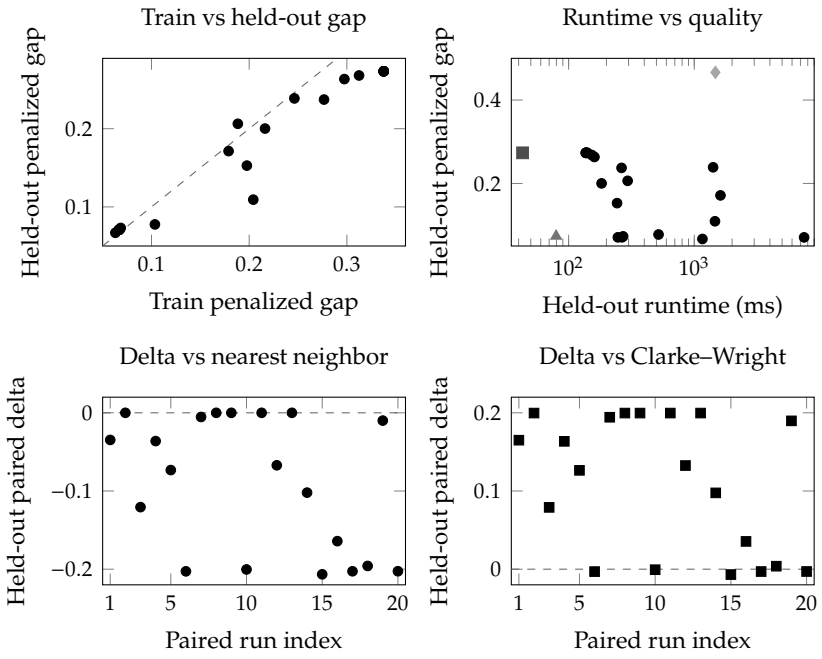

Figure 4.3: Diagnostic views for the official phase-9 solver-evolution campaign, using all twenty paired runs. The upper panels show that good train performance does not guarantee strong held-out performance and that runtime has a heavy upper tail. The lower panels show the paired held-out deltas against the two most important baselines. Negative values are better because the endpoint is lower-is-better penalized gap. In the runtime panel, circular markers denote solver-evolution runs, while square, triangular, and diamond markers denote nearest-neighbor, Clarke–Wright, and regret-insertion-plus-local-search baselines.

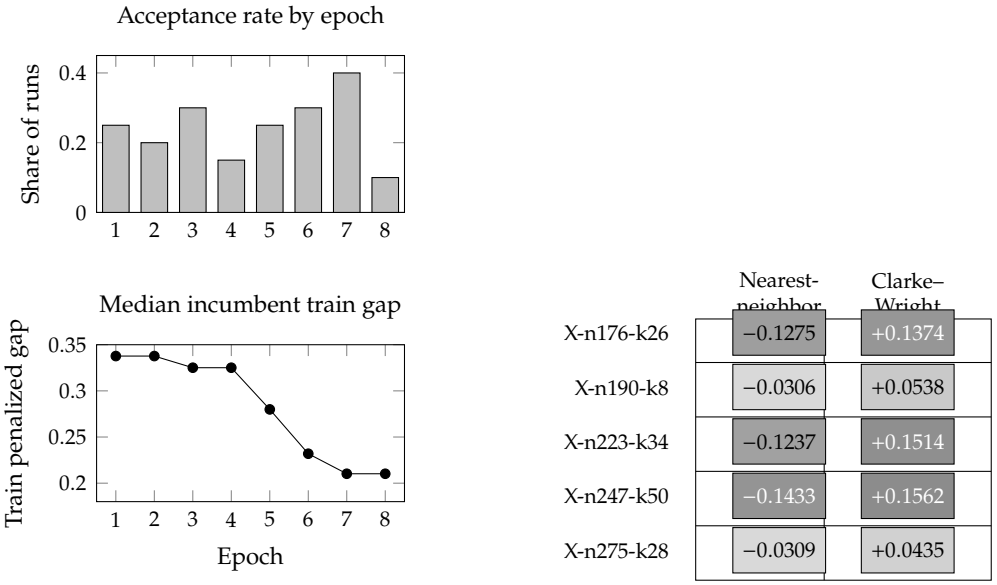

Figure 4.4: Additional diagnostics for the official phase-9 solver-evolution campaign. The left panels use all twenty official runs and track the accepted-update rate by epoch together with the median incumbent train penalized gap, starting from the shared nearest-neighbor constructive incumbent. The right panel aggregates the five held-out Uchoa X-series instances across the same twenty runs. Solver evolution has lower gap than nearest-neighbor on every held-out instance and higher gap than Clarke–Wright on every held-out instance. Darker cells mark larger absolute differences, and negative values favor solver evolution.

Condition	Held-out feasibility	Mean penalized gap	Reading
Clarke–Wright savings	1.0	0.073766	strongest fixed baseline, unchanged from the original phase-9 suite
Replay solver evolution	1.0	0.171114	only learned arm with full held-out feasibility under the closeout controls
Nearest-neighbor constructive	1.0	0.273435	weaker fixed baseline retained for continuity with phase 9
Regret insertion plus local search	1.0	0.466391	weakest fixed baseline on the primary endpoint
Budget-matched no replay	0.15	2.932746	memoryless learned control with the same overall candidate budget
Direct generate plus one repair	0.25	5.286402	bounded direct-synthesis control with one scripted repair step

Table 4.9: Held-out results for the official phase-9 closeout campaign, based on twenty paired runs over the same five held-out Uchoa X-series instances used in the original phase-9 suite. Lower penalized gap is better.

Appendix A.5 shows a representative accepted phase-9 solver fragment, and Appendix A.6 records a representative validator rejection. Those artifacts help explain both sides of the result: the loop can preserve feasible route-improvement logic, but validator pressure still filters out many large or syntactically salvaged candidates.

4.4.4 Phase 9 Closeout: Budget-Control CVRP Study

The original phase-9 campaign established that bounded solver evolution on real-world CVRP is feasible and stronger than the weakest fixed baselines. It left one mechanism question open: was replay helping because it preserved useful incumbent structure, or because the loop was allowed to spend more candidate budget than a direct generation baseline? The closeout keeps the same bounded Uchoa X-series CVRPLIB split, validator, and held-out endpoint, then introduces matched learned controls to answer that narrower question directly.

The official closeout campaign contains all six conditions in every paired run, judge support remains enabled throughout, and the three learned closeout arms record zero generation fallbacks and zero generation errors. Replay-aware solver evolution is also the only learned arm with full held-out feasibility in all twenty official runs. The budget-matched no-replay arm remains feasible in only three runs, and the direct-generate-plus-one-repair arm remains feasible in only five. Table 4.9 summarizes the held-out outcome.

The paired closeout comparisons quantify the mechanism difference more directly. Replay-aware solver evolution has a held-out penalized gap that is lower than the budget-matched no-replay control by 2.761632 penalized-gap units. The paired 95%

CI is [−3.369777, −2.201586]. It is also lower than the direct-generate-plus-one-repair control by 5.115288 penalized-gap units, with paired 95% CI [−7.451608, −2.946446]. Against Clarke–Wright, the sign remains unfavorable: replay-aware solver evolution is higher by 0.097348 penalized-gap units, with paired 95% CI [0.055994, 0.138558].

The closeout addresses the mechanism question because the weaker learned arms fail in different ways. The budget-matched no-replay arm accepts more epochs than replay-aware solver evolution, yet it often produces code changes that do not preserve held-out feasibility. The bounded direct control produces a competitive feasible solver in a few runs and fails validation in most runs on the same held-out panel. The contrasts indicate that replay contributes more than extra candidate count alone: in this bounded regime it stabilizes the search around an incumbent well enough to produce lower penalized gaps than both learned controls, even though it remains behind the strongest fixed Clarke–Wright baseline.

4.4.5 Phase 9 Closeout Diagnostic: Feasibility and Route Quality

The large budget-matched no-replay gap is mainly a feasibility artifact. The held-out audit found that the budget-matched no-replay arm was fully feasible in only three of twenty paired runs. Seventeen runs returned semantically invalid solver outputs, most commonly outputs that were not lists of routes. Among the three feasible no-replay runs, two were close to the Clarke–Wright level, with gaps near 0.0708, while one run had a severe feasible-route gap of 7.513251. The direct-generate-plus-one-repair arm shows a related but distinct pattern: it was feasible in five of twenty runs and those feasible runs had mean gap about 0.073607, while the other fifteen runs failed validation. The closeout therefore supports a reliability claim. The replay-aware incumbent-preserving loop is much more reliable than memoryless learned controls at maintaining feasibility under repeated code edits.

4.4.6 External HGS-Style CVRP Calibration

To calibrate the phase-9 CVRP results against a stronger specialized solver, the phase-12 external baseline was run on the same five held-out Uchoa X-series CVRPLIB instances using PyVRP 0.13.4, an open-source solver package implementing hybrid genetic search ideas for vehicle routing [33, 34]. The calibration used five fixed seeds per instance and a 30-second runtime cap per seed. Each returned solution was revalidated with the same project validator used for the phase-9 closeout, including the same capacity and customer-coverage checks.

The calibration bounds the interpretation of the phase-9 solver results. PyVRP's best-seed mean held-out gap of 0.006197 is lower than the Clarke–Wright mean gap of 0.073766 and far lower than the closeout replay-evolution mean gap of 0.171114. The calibration does not invalidate the phase-9 mechanism claim, because PyVRP is an external specialized solver outside the learned code-evolution arms. It does rule

Instance	Best seed	Cost	Best known	Gap	Wall seconds
X-n176-k26	5	48161	47812	0.007299	30.036
X-n190-k8	4	17020	16980	0.002356	30.042
X-n223-k34	4	40726	40437	0.007147	30.056
X-n247-k50	5	37671	37274	0.010651	30.057
X-n275-k28	3	21320	21245	0.003530	30.058
Mean	–	–	–	0.006197	30.050

Table 4.10: PyVRP HGS-style calibration on the same phase-9 held-out Uchoa X-series CVRPLIB instances. The table reports the best validated seed per instance from five fixed seeds with a 30-second runtime cap per seed.

out any practical claim that the evolved CVRP solvers are competitive with modern specialized VRP search on this benchmark slice.

4.4.7 Interpretive Pattern Across the Later Phases

Phases 7 through the phase-9 closeout share a common pattern. As the search object or validator becomes more demanding, the loop continues to produce executable diversity and the surviving positive claims become narrower.

Phase 7 produced rediscovery without validated reusable operator discovery. Phase 8 improved over a weak one-shot LLM baseline while remaining below the strongest fixed or supervised selectors when selector headroom was absent. Phase 9 produced feasible real-world solver evolution with lower gap than weak baselines and higher gap than the best fixed baseline. The phase-9 closeout then separated replay-aware incumbent preservation from matched learned controls, although the strongest fixed classical baseline remained better.

Taken as a whole, these later phases motivate the dissertation's capability-bounded interpretation. The evidence includes non-zero held-out improvement under some conditions. The later phases do not support the stronger claim that the system discovers dominant general-purpose optimization methods.

4.5 Cross-Model Budget-Control Factorial

The final crossed control study tests whether the retained replay effects survive a three-model, budget-controlled comparison. The archived campaign contains 750 analyzed rows across simple games, symmetric TSP, and real-world CVRP, three model tiers, and five generation techniques: single-shot generation, budget-matched no-replay search, random replay, failure replay, and failure replay with compression. The model ladder was the budget-feasible ladder used in the official run, namely `gpt-4.1-nano`, `gpt-5-nano`, and `gpt-5.4-mini`. The analysis used the same paired-by-seed and bootstrap discipline used elsewhere in the thesis.

The factorial gives a negative control outcome. Replay, failure replay, and compressed failure replay had lower score than the budget-matched no-replay control in zero task-by-model comparisons with positive bootstrap support. The pooled continuous variance partition attributed only $R^2 = 0.000884$ to model strength, $R^2 = 0.010015$ to evolution technique, and $R^2 = 0.069086$ to their interaction, with residual variance 0.829829. Here, R^2 denotes the share of variance attributed to each factor in the variance partition. Task-specific model-strength contributions were also small: $R^2 = 0.007179$ for simple games, $R^2 = 0$ for TSP, and $R^2 = 0.002209$ for CVRP. These values do not support a claim that base-model strength alone explains the dissertation results.

For the hypotheses, the factorial result narrows the interpretation without reversing it. H1 remains partially supported because executable adaptation occurs in earlier controlled phases, but H2 is not supported as a universal replay claim: replay-specific mechanisms must clear the budget-matched control, and this final factorial did not do so. H3 is strengthened because the budget control and stronger baselines reduce the apparent advantage of the evolved variants. H4 is also strengthened because the results differ by task family and do not collapse to a single model-strength explanation.

4.6 Failure-Mode Synthesis

Table 4.11 consolidates the recurring failure modes observed across the dissertation. It complements the statistical endpoints by explaining why some apparently promising search variants do not support stronger claims.

Failure mode	Where it appears	Empirical symptom	Interpretation
Lexical churn	simple-game novelty audit	large code changes without stable held-out improvement	Novelty must be adjudicated against behavior and held-out performance, not token difference alone.
Validator gaming or brittle semantics	simple games and generated routing code	high apparent novelty or partial success that fails stricter checks	Automated validators need independent held-out panels and qualitative artifact review.
Weak transfer across families	phase-5 TSP, ATSP, and CVRP	replay gains on symmetric TSP do not reproduce uniformly on ATSP or CVRP	Replay is a mechanism whose effect depends on benchmark structure.
Rediscovery without reusable discovery	phase-7 operator search	repeated double-bridge or segment-reversal signatures without transplant success	Recognizable operator families are insufficient unless reuse is validated.
Portfolio headroom exhaustion	phase-8 adaptive control	adaptive controller has lower gap than a weak LLM selector and higher gap than the strongest fixed heuristic	If the frozen portfolio contains little oracle headroom, controller search has limited room to improve.
Feasibility collapse	phase-9 closeout learned controls	budget-matched no-replay feasibility 0.15 and direct-plus-repair feasibility 0.25	The closeout advantage of replay is mainly stability under validator pressure.
Specialized-solver gap	PyVRP HGS-style calibration	PyVRP best-seed mean held-out gap 0.006197 versus closeout replay 0.171114	Current evolved solvers remain research artifacts relative to mature VRP solvers.

Table 4.11: Failure-mode synthesis across the retained empirical campaigns.

Chapter 5

Conclusions and Future Work

The central question was when an LLM-guided code-evolution loop can generate useful heuristic adaptation and when apparent gains are better explained by search budget, benchmark headroom, validator pressure, or baseline strength. The answer is conditional. The loop can generate executable variation and sometimes preserve held-out improvements. Stronger claims fail under the full set of controls. Replay helps in selected regimes, especially in the phase-9 closeout where it stabilizes feasible solver evolution against learned controls. The evidence does not support replay as a universal mechanism, and mature classical and hybrid solvers remain stronger on the hardest CVRP calibration.

5.1 Assessment of the Empirical Hypotheses

Table 5.1 summarizes the final assessment of the hypotheses stated in Section 1.3. The assessment uses the full evidence base in Table 4.1, including the phase-9 closeout, the PyVRP HGS-style calibration, and the three-model budget-control factorial.

5.2 Answer to the Central Question

The central question has a conditional answer. An LLM-guided code-evolution loop is most useful when the task has reachable heuristic headroom, deterministic validation is available, the search object remains small enough for repeated executable edits, and the baseline does not already consume most of the available structure. In those regimes, the loop can produce useful adaptation and concrete artifacts for inspection.

Apparent gains are better explained by search budget, benchmark headroom, validator pressure, or baseline strength when they disappear under budget-matched controls, fail to transfer across related benchmark families, depend on infeasible or brittle artifacts, or remain far behind specialized solvers. The final CVRP

Hyp.	Assessment	Evidence
H1	Partially supported	The loop produced functional adaptation in simple games, a statistically separated symmetric-TSP replay result, and feasible CVRP solver variants. The support is partial because gains are not consistent across all families or baselines.
H2	Locally supported only	Replay must clear budget-matched controls. It does so in the phase-9 closeout against learned controls. The three-model factorial found zero replay/failure/compression advantages over budget-matched no replay with positive bootstrap support.
H3	Supported	Gains shrink as validators and baselines strengthen. Phase 8 is constrained by low portfolio headroom, phase 9 remains behind Clarke–Wright, and the PyVRP HGS-style calibration produces lower gaps than the evolved CVRP solvers.
H4	Supported	Transfer is partial. The TSP replay result does not transfer uniformly to ATSP or CVRP, and the model-strength factorial shows task-specific interactions without reducing to one general replay or model-strength effect.

Table 5.1: Final assessment of the empirical hypotheses after the closeout, external-baseline, and model-factorial analyses.

evidence illustrates both sides. Replay-aware solver evolution is valuable relative to memoryless learned controls because it preserves feasibility under repeated edits. The PyVRP calibration bounds that value against a much stronger vehicle-routing reference point.

5.3 Limitations

Internal validity. The main internal-validity protection is the use of paired offsets, deterministic validators, held-out panels, aggregate reports, and technical audits. Remaining risk comes from implementation choices in the validators, finite seed counts, and the possibility that some generated-code failures reflect details of the sandbox rather than the underlying search concept.

External validity. The benchmark coverage compares simple games, TSP, ATSP, and CVRP without surveying all combinatorial optimization. The CVRP experiments use a bounded Uchoa X-series CVRPLIB subset. The conclusions should transfer only to settings with similar executable-code interfaces, feedback loops, and validation constraints.

Construct validity. The thesis measures held-out score, relative gap, penalized gap, feasibility, novelty diagnostics, and selected artifact-level failure modes. These constructs capture useful dimensions of heuristic adaptation. They do not exhaust algorithmic creativity, maintainability, or human usefulness. Novelty in particular is only diagnostic unless it is tied to held-out performance or validated behavioral change.

Conclusion validity. The dissertation reports confidence intervals and budget-matched controls where the design supports them, but some early game evidence remains more qualitative than the later routing evidence. The strongest conclusions are comparative and bounded. Replay-aware evolution can stabilize learned CVRP code against learned controls, but the evidence does not show a universal replay advantage or competitiveness with modern specialized VRP solvers.

5.4 Future Work

Future work should treat this thesis as a measurement foundation for further algorithm-discovery studies. A broader model-strength continuum could extend the completed three-tier factorial with more models and tighter budget normalization. Larger CVRP or VRPTW studies could test whether the feasibility-preservation result survives richer constraints and stronger external baselines. Another direction is human-in-the-loop solver design, where the model proposes bounded code edits and a human researcher selects or revises interpretable components. Finally, the artifact appendix suggests a need for better measures of functional novelty that combine code structure, behavior, and transfer without relying on lexical distance alone.

5.5 Final Conclusion

The evidence gives a capability-bounded view of LLM-guided code evolution. The loop can generate executable heuristic variation, preserve useful adaptations in selected settings, and expose informative failure modes. It is not yet a general-purpose autonomous algorithm-discovery system. The dissertation makes these boundaries measurable: useful adaptation can be separated from code churn, replay-specific value from extra candidate budget, and research-stage solver evolution from the performance of mature optimization methods.

Bibliography

[1] Rishabh Agarwal et al. "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *Advances in Neural Information Processing Systems 35*. 2021.

[2] Mohamad Alissa, Ender Özcan, and Graham Kendall. "Automated Algorithm Selection: from Feature-Based to Feature-Free Approaches". In: *Journal of Heuristics* 29.1 (2023), pp. 1–38.

[3] Jacob Austin et al. "Program Synthesis with Large Language Models". In: *arXiv preprint arXiv:2108.07732* (2021).

[4] Irwan Bello et al. "Neural Combinatorial Optimization with Reinforcement Learning". In: *arXiv preprint arXiv:1611.09940* (2016).

[5] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine Learning for Combinatorial Optimization: A Methodological Tour d'Horizon". In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421.

[6] Yoshua Bengio et al. "Curriculum Learning". In: *Proceedings of the 26th International Conference on Machine Learning*. 2009, pp. 41–48.

[7] Marc Brittain et al. "Prioritized Sequence Experience Replay". In: *arXiv preprint arXiv:1905.12726* (2019).

[8] Edmund K. Burke et al. "Hyper-heuristics: A Survey of the State of the Art". In: *Journal of the Operational Research Society* 64.12 (2013), pp. 1695–1724.

[9] Mark Chen et al. "Evaluating Large Language Models Trained on Code". In: *arXiv preprint arXiv:2107.03374* (2021).

[10] Geoff Clarke and John W. Wright. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". In: *Operations Research* 12.4 (1964), pp. 568–581.

[11] Hanjun Dai et al. "Learning Combinatorial Optimization Algorithms over Graphs". In: *Advances in Neural Information Processing Systems 30*. 2017.

[12] George Dantzig, Ray Fulkerson, and Selmer Johnson. "Solution of a Large-Scale Traveling-Salesman Problem". In: *Journal of the Operations Research Society of America* 2.4 (1954), pp. 393–410.

[13] Tansel Dökeroglu, Tayfun Kucukyilmaz, and El-Ghazali Talbi. "Hyper-heuristics: A Survey and Taxonomy". In: *Computers & Industrial Engineering* 187 (2024), p. 109815.

[14] Alex Graves et al. "Automated Curriculum Learning for Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 1311–1320.

[15] Keld Helsgaun. "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic". In: *European Journal of Operational Research* 126.1 (2000), pp. 106–130.

[16] Peter Henderson et al. "Deep Reinforcement Learning That Matters". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (2018).

[17] Naman Jain et al. "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code". In: *arXiv preprint arXiv:2403.07974* (2024).

[18] Juyong Jiang et al. "A Survey on Large Language Models for Code Generation". In: *arXiv preprint arXiv:2406.00515* (2024).

[19] Yujia Li et al. "Competition-Level Code Generation with AlphaCode". In: *Science* 378.6624 (2022), pp. 1092–1097.

[20] Shen Lin and Brian W. Kernighan. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". In: *Operations Research* 21.2 (1973), pp. 498–516.

[21] Shengcai Liu et al. "Large Language Models as Evolutionary Optimizers". In: *2024 IEEE Congress on Evolutionary Computation (CEC)*. 2024, pp. 1–8.

[22] Daniel J. Mankowitz et al. "Faster Sorting Algorithms Discovered Using Deep Reinforcement Learning". In: *Nature* 618 (2023), pp. 257–263.

[23] Jean-Baptiste Mouret and Jeff Clune. "Illuminating Search Spaces by Mapping Elites". In: *arXiv preprint arXiv:1504.04909* (2015).

[24] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. "Quality Diversity: A New Frontier for Evolutionary Computation". In: *Frontiers in Robotics and AI* 3 (2016), p. 40.

[25] Gerhard Reinelt. "TSPLIB—A Traveling Salesman Problem Library". In: *ORSA Journal on Computing* 3.4 (1991), pp. 376–384.

[26] John R. Rice. "The Algorithm Selection Problem". In: *Advances in Computers* 15 (1976), pp. 65–118.

[27] Bernardino Romera-Paredes et al. "Mathematical Discoveries from Program Search with Large Language Models". In: *Nature* 625 (2024), pp. 468–475.

[28] Francesca Da Ros et al. "Large Language Models for Combinatorial Optimization: A Systematic Review". In: *arXiv preprint arXiv:2507.03637* (2025).

[29] Tom Schaul et al. "Prioritized Experience Replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[30] David Silver et al. "Mastering the Game of Go without Human Knowledge". In: *Nature* 550 (2017), pp. 354–359.

[31] Petru Soviany et al. "Curriculum Learning: A Survey". In: *International Journal of Computer Vision* 130 (2022), pp. 1526–1565.

[32] Eduardo Uchoa et al. "New Benchmark Instances for the Capacitated Vehicle Routing Problem". In: *European Journal of Operational Research* 257.3 (2017), pp. 845–858.

[33] Thibaut Vidal. "Hybrid Genetic Search for the CVRP: Open-Source Implementation and SWAP* Neighborhood". In: *Computers & Operations Research* 140 (2022), p. 105643.

[34] Niels A. Wouda and Leon Lan. "PyVRP: A High-Performance VRP Solver Package". In: *INFORMS Journal on Computing* 36.4 (2024), pp. 943–955.

[35] Daoguang Zan et al. "Large Language Models Meet NL2Code: A Survey". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2023, pp. 7443–7464.

Appendix A

Representative Evolved Artifacts and Failure Modes

This appendix presents short excerpts from archived artifacts and connects aggregate claims to concrete generated code. The excerpts are intentionally brief. They show the kinds of changes the loop actually produced, without replacing the aggregate evidence with anecdotes.

A.1 Successful Simple-Game Adaptation

Listing A.1 comes from a replay-aware pursuit/evasion run that was retained as a functional adaptation in the phase-4 novelty audit. Its structural change is the opponent-model helper `opp_best_after`, which scores each candidate move against the opponent's likely reply and against the opponent's current position.

Listing A.1: Accepted pursuit/evasion policy excerpt from the replay-aware transfer suite.

```
def opp_best_after(selfx, selfy):
    opp_opts = neighbors(ox, oy)
    if not opp_opts:
        return ox, oy
    best = None
    bestv = None
    for px, py in opp_opts:
        v = d2(selfx, selfy, px, py)
        if opp_evader:
            if best is None or v > bestv:
                best, bestv = (px, py), v
        else:
            if best is None or v < bestv:
                best, bestv = (px, py), v
    return best

```

```
if self_evader:
    for nx, ny in self_moves:
        px, py = opp_best_after(nx, ny)
        v = d2(nx, ny, px, py)
        if bestv is None or v > bestv:
            bestv, best_move = v, (nx, ny)
```

A.2 Novel but Unhelpful Churn

Listing A.2 is a representative novelty-spike artifact from the replay-aware territory-control audit. The artifact is structurally coherent and visibly different from the incumbent. The archived selection record marks it as rejected because its score regressed from 50.5 to 39.5 against the same opponent.

Listing A.2: Rejected territory-control novelty spike from the replay-aware transfer audit.

```
if (nx, ny) in ot:
    sc += 18.0
elif (nx, ny) in unq:
    sc += 10.0
elif (nx, ny) in st:
    sc += 6.0
else:
    sc += 4.0

dsc = man(nx, ny, cx, cy)
sc += -0.7 * dsc
sc += 0.35 * man(ox, oy, cx, cy)

dso = man(nx, ny, ox, oy)
sc += -0.08 * dso
```

A.3 Replay-Generated TSP Heuristic Fragment

Listing A.3 shows a replay-generated TSP heuristic fragment from the phase-5 symmetric TSPLIB suite. The evolved object is a bounded configuration over construction, local-search, perturbation, restart, and acceptance modules.

Listing A.3: Phase-5 replay-generated TSP heuristic fragment.

```
def build_heuristic():
    return {
        "construction": {
            "seed_mode": "nearest_centroid",
            "candidate_limit": 28,
            "lookahead_limit": 4,
            "distance_weight": 1.05,
```

```
            "regret_bonus": 0.45,
        },
        "local_search": {
            "two_opt_passes": 8,
            "use_three_opt": True,
            "three_opt_samples": 10,
        },
        "perturbation": {
            "enabled": True,
            "mode": "segment_reversal",
            "strength": 2,
        },
        "acceptance": {
            "mode": "annealed",
            "worse_acceptance_threshold": 0.012,
        },
    }
```

A.4 Rediscovered Operator Family

Listing A.4 is a phase-7 perturbation operator that exemplifies the double-bridge and segment-reversal family discussed in the rediscovery analysis. The phase-7 evidence shows repeated rediscovery of related perturbation families, although this specific operator did not survive as a reusable component.

Listing A.4: Phase-7 perturbation operator from the rediscovered double-bridge family.

```
def build_operator():
    return {
        "name": "perturbation_two_cluster_bottleneck_escape_double_bridge_compact",
        "description": "Deterministic escape schedule for 2-opt stagnation.",
        "operator_type": "perturbation",
        "primary_mode": "double_bridge",
        "secondary_mode": "segment_reversal",
        "stagnation_threshold": 3,
        "strength": 3,
        "attempts": 2,
        "escalation_strength": 2,
    }
```

A.5 Bounded CVRP Solver Evolution

Listing A.5 shows a short excerpt from an accepted phase-9 CVRP solver. The fragment is representative of the later successful solvers: a deterministic Clarke–Wright construction remains visible, followed by bounded route-improvement logic instead of a simple constructive pass alone.

Listing A.5: Accepted phase-9 CVRP solver fragment.

```
def improve_swap(rs):
    if len(rs) < 2:
        return False
    best_delta = 0
    best_move = None

    for i in range(R - 1):
        ri = rs[i]
        for j in range(i + 1, R):
            rj = rs[j]
            for p in range(len(ri)):
                a = ri[p]
                for q in range(len(rj)):
                    b = rj[q]
                    if li - demands[a] + demands[b] > capacity:
                        continue
                    if lj - demands[b] + demands[a] > capacity:
                        continue
                    new_i = ri[:]
                    new_j = rj[:]
                    new_i[p] = b
                    new_j[q] = a
```

A.6 Rejected Validator Case

The rejected cases show how validator pressure shaped the analyzed artifacts. Listing A.6 records a phase-9 rejection in which a large candidate was salvaged syntactically but still violated the solver-output contract. The example shows why the later chapters require executable validation before a generated solver can enter the analysis.

Listing A.6: Phase-9 rejection record with validator failure.

```
"accepted": false,
"errors": [
  "solver did not return a list of routes"
],
"generation_fallback_used": false,
"salvage_attempted": true,
"validation_issues": [
  "salvaged by trimming incomplete trailing block"
]
```