

POLITECNICO
DI TORINO

this: POLITECNICO DI TORINO

Master's Degree in ICT for Smart Societies

Thesis

Study whether an LLM can improve the training of a small NN for indoor localization, compared to conventional optimization methods

Supervisor
Prof. Mihai Lazarescu

Candidate
Hossein Favakeh

Academic Year 2025-2026

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Mihai Lazarescu, for his invaluable guidance throughout this thesis. He consistently encouraged me to understand how systems work and, more importantly, to be able to explain the reasoning behind what I presented rather than focusing on the final results. This discipline shaped the way I approached the entire project, including the way I used AI tools. I am deeply grateful for his patience, rigor, and continuous support.

I am also grateful to my family for their unwavering love and support. Leaving Iran to pursue my education meant facing the challenges of immigration and carrying the weight of distance and uncertainty. Throughout all of this, their constant encouragement and belief in me remained a source of strength and gave me the resilience to continue this journey.

AI use disclosure

Generative artificial-intelligence tools were used in the preparation of this thesis. This declaration states which tools, for what, and where.

Tools: Anthropic's Claude, accessed through Claude Code, OpenAI's ChatGPT, Perplexity and, SpaceXAI's Grok; these were used as a working aid.

Code and analysis: The assistants were used for software work: writing and debugging the experiment pipeline, analysis scripts, plotting and, cross-checking the values quoted in the tables against the stored run logs. Explaining the logic of the code, formulas, and understanding of concepts.

Text: The assistant was used for language correction in an academic way. Also, get help to frame and structure the report in LaTeX.

Not used for: No part of the literature search or citation selection was delegated to a generative tool, and no reference in the bibliography was produced by one.

Abstract

Indoor localization underpins services such as assisted living, occupancy-aware building control, and access management. This thesis studies how to improve person-movement estimation using a small Long Short-Term Memory (LSTM) network with the help of an agent LLM. The accuracy of the network depends on its hyperparameters, and tuning them is costly because every candidate configuration must be trained from scratch. The model is given a qualitative summary of the trials so far rather than the raw measured numbers, and, in the second half of the study, a description of how the tracked person moves.

An agent LLM is proposed. At each round, it reads a compact qualitative summary of the trials so far and proposes the next configuration, subject to strict validation that rejects any out-of-grid, malformed, or repeated proposal, so that it reaches only configurations also available to the other methods. It is evaluated under an identical protocol against a fixed baseline, random search, a rule-based controller, and Optuna. Configurations are scored by the mean validation RMSE (in meters) over three seeds, and each method's selection is re-evaluated on unused seeds with paired statistics, measuring not only final accuracy but also reliability, how the attempt budget is spent, and how quickly the score drops.

Across all complete search runs over four open models, the LLM did not improve on the conventional methods: it averaged 0.2455 m test RMSE against 0.2413 m for random search and 0.2371 m for the rule-based controller, and in this comparison, the manually tuned baseline has the lowest RMSE(m) at 0.2308 m. Diagnostic experiments trace this to the information the LLM receives rather than to the search strategy. Later on, by adding a per-epoch training-curve summary to the rule-based controller reaches 0.2264 m, whereas removing the trial history improves the LLM's test error instead of harming it, while the validation error it actually minimizes gets worse. That gain is not evidence of a freer search: with the history removed the model evaluates fewer distinct configurations, covers less of the grid, and wastes half its budget re-proposing settings it has already tried. It also does not hold for every model, one of five being harmed by the same change.

A second experiment supplies what a language model might uniquely know. With the architecture frozen and human-motion descriptors provided, the model reshapes the training loss instead of the hyperparameters, and here it does improve on baseline, reaching 0.2293 m against 0.2308 m; withholding the motion summary removes that gain entirely, so the motion information demonstrably affects what the model chooses. The advantage is again marginal, however: a rule-based reading the same descriptors reaches 0.2283 m beats baseline, and random search of the same six loss parameters is statistically indistinguishable from the model. On a second, infrared recording the effect does not contract but reverses: both methods that read the motion descriptors fall behind baseline training, while random search remains the only method to improve on it. The defensible conclusion is therefore neither that the model exploits human-motion knowledge nor that such knowledge has no effect, but that motion information influences and

can improve the model's choices on one recording, without evidence that it is used better than by random search or a fixed heuristic, and without transferring to the second sensor. What information the model is given matters more than the fact that it reasons over that information.

Contents

1 Introduction 7
- 1.1 Context and Background . . . 7
- 1.2 Motivation . . . 7
- 1.3 Problem Statement and Research Question . . . 8
- 1.4 Contributions . . . 9
- 1.5 Thesis Organization . . . 10

2 Background and Related Work 11
- 2.1 Indoor Localization . . . 11
- 2.2 Hyperparameter Optimization . . . 12
  - 2.2.1 Grid and Random Search . . . 12
  - 2.2.2 Bayesian Optimization and Optuna . . . 12
  - 2.2.3 Overtuning and Validation Overfitting . . . 13
- 2.3 Large Language Models as Optimizers . . . 13
- 2.4 Domain Knowledge and Human Motion . . . 14
- 2.5 Related Work . . . 15

3 LLM-Driven Hyperparameter Optimization 17
- 3.1 Overview of the Approach . . . 17
- 3.2 Search Space and Scoring . . . 17
- 3.3 Agent . . . 19
- 3.4 Strict Validation of Proposals . . . 19
- 3.5 Conventional Methods . . . 20
  - 3.5.1 Fixed Baseline . . . 20
  - 3.5.2 Random Search . . . 21
  - 3.5.3 Rule-Based Search . . . 21
  - 3.5.4 Optuna (Bayesian Optimization) . . . 21
- 3.6 Extending the Training Objective . . . 21

4 Experimental Evaluation 23
- 4.1 Methodology . . . 23
  - 4.1.1 Dataset and Preprocessing . . . 23
  - 4.1.2 Localizer and Training . . . 24
  - 4.1.3 Search Protocol and Scoring . . . 25

4.1.4 Methods Compared and Implementation . . . 26
4.1.5 Evaluation and Metrics . . . 26
4.2 Numerical Results . . . 27
4.2.1 Final Accuracy . . . 27
4.2.2 Reliability . . . 29
4.2.3 How the Budget Is Spent . . . 29
4.2.4 Optimization Dynamics . . . 31
4.3 Why the LLM Fails . . . 31
4.3.1 Per-Epoch Curve Summaries . . . 31
4.3.2 Does the LLM Use the History? . . . 31
4.3.3 Prompt and Model Capacity . . . 34
4.3.4 How the History Is Represented . . . 35
4.3.5 Where the LLM Improves, Degrades and Stalls . . . 37
4.3.6 Selection Against a Noisy Validation Split . . . 38

5 Motion-Aware Optimization 41
5.1 Motion-Aware Objective . . . 41
5.1.1 Units, Shapes and Normalisation . . . 42
5.1.2 Why this Smoothness Prior Is Misspecified . . . 44
5.2 Results . . . 45
5.2.1 Arms and Controls . . . 45
5.2.2 Radar . . . 46
5.2.3 Where the Gain Comes From . . . 48
5.2.4 A Second Sensor: IR . . . 49
5.2.5 The Smoothness Prior in Isolation . . . 51
5.2.6 Run Integrity . . . 54
5.2.7 Does the LLM Reason, or Label? . . . 54
5.3 What the Motion Experiment Establishes . . . 55

6 Conclusion 57
6.1 Limitations . . . 58
6.2 Future Work . . . 58

A Manual Tuning of the Baseline Configuration 61

B The Localizer Network 65

C The Agent: Prompts, Inputs and Outputs 67
C.1 The system prompt . . . 67
C.2 The user message . . . 68
C.3 Validation, feedback and the retry . . . 70
C.4 The prompt after motion knowledge . . . 70

D The Rule-Based Controller 75
D.1 The motion controller . . . 77

Bibliography 79

Chapter 1

Introduction

1.1 Context and Background

Tracking person movement inside a building is the starting point for many useful services. Examples include guiding visitors through large public spaces, monitoring the well-being of a person in a room, and heating and lighting only the occupied rooms.

A practical indoor solution should be inexpensive, small, should not require people to carry or wear any device, and also respect their privacy. A low-cost sensor mounted in the environment meets these requirements well. We worked with a radar, low-resolution IR. For each frame, they report numbers, angle, and range (distance) of the detected target, together with a measure of the reflected signal strength, sampled four times per second for the radar and five for the infrared array. It captures enough information to follow a moving target. The data is a stream of numbers that changes over time as the person walks.

Turning these time-series numbers into an actual position is the role of a machine learning model. The model receives a short window of recent sensor readings and predicts the two coordinates, X and Y, of the person on the floor plan. For the time-series data, we used a Long Short-Term Memory (LSTM) network. The model is kept small so that it can run on local devices (laptop) rather than on a powerful computer or in the cloud. Keeping the computation on the device reduces cost and latency and limits the amount of data that leaves the room.

1.2 Motivation

Tuning a small neural network takes time and effort. Its behavior depends on a set of design choices that are fixed before training begins. Examples are how many time steps the input window covers, how many memory units the network has, how strongly it is regularized, and how fast it learns.

In practice, machine learning experts use conventional strategies such as preprocessing and cleaning the data, feature extraction, model selection, hyperparameter optimization, and analysis of the model and its results. More systematic options are random search, which samples configurations randomly, and Bayesian optimization, which uses past trials

to decide what to try next and is implemented in widely used frameworks such as Optuna. A further option is a rule-based search that acts on the label, which is deterministic. Each of these methods has known strengths and weaknesses in terms of accuracy, the number of trials needed, and the total computing time spent.

Recently, large language models (LLMs) were able to automate tasks and follow structured rules described in plain text. This assumption comes to machine learning that manuallconfiguresng orfollowsg a fixed statistical rule; an LLM could read past trials, reason about them, and propose the next configuration to try. With this optimization, it might reach optimal values in fewer trials. The potential benefit is that it can reason mmulti-objectivelyto set configurations and learn from history; the other conventional methods cannot.

1.3 Problem Statement and Research Question

This thesis investigates questions as follows:

1. Can a large language model improve the optimization of a small LSTM for indoor localization compared with conventional methods?
2. Does the LLM performance depend on the representation of the history provided in context?
3. Does the LLM use the outcome information in the search history, or can similar behavior be obtained without access to the true history?
4. Can human-motion knowledge fed to an LLM improve the training objective for indoor localization beyond the flexibility provided by additional tunable loss parameters?

To make the question answerable, the methods are evaluated using five criteria: every method ranks candidate configurations on a single objective, the mean validation RMSE in meters over three fixed training seeds.

- **Final accuracy**: how close the predicted position is to the true position once the best configuration found by a method is fully trained. Measured as test RMSE in meters, with the selected configuration retrained from scratch on 30 fresh seeds.
- **Reliability**: whether a method's selection beats the baseline repeatedly, then another one succeeds less than fails that is not useful in practice. We measured across the same fresh seeds the spread of test RMSE and the number of seeds on which one selection is the better of the pair.
- **Sample efficiency**: how a method spends a fixed budget of 25 attempts. How many attempts become real evaluations rather than being consumed by proposals that are never trained, how much of the search space is covered, as both the count of distinct configurations evaluated and the fraction of allowed values selected per hyperparameter, and how many attempts pass before the best of them is found.

- **Optimization dynamics**: how fast the score is driven down over the budget, which distinguishes a method that converges early from one that is still improving when the budget ends.
- **Proposal validity**: what fraction of a method's proposals are usable at all. Unique to the LLM arm, which alone can provide a malformed outcome, an out-of-grid value, a repeat of an already evaluated configuration, or no response within the timeout. Since a rejected proposal still consumes an attempt without training, this criterion converts the model's output quality into a measurable cost against a budget shared equally by all methods.

The comparison is controlled. An agent LLM is placed in competition with a fixed baseline, random search, a rule-based method, and Optuna. Every method works on the same dataset, the same LSTM, the same shared search space, and the same fixed budget, so the protocol isolates differences arising from how each method selects the next configuration, given the information supplied to it.

1.4 Contributions

The main contributions of this work are the following:

- An agent LLM for a small LSTM localizer, in which the model proposes configurations from the history of past trials under strict validation of its outputs.
- A controlled benchmark that compares this agent against four conventional methods under an identical protocol, so that the comparison is fair.
- An evaluation that goes beyond final accuracy to also report reliability, sample efficiency and proposal validity, including a final stage in which each method's best configuration is re-evaluated on a set of fresh seeds and compared to the LLM with paired statistics.
- A measurement of how the LLM actually spends its budget, how many of its proposals survive validation, and how many distinct configurations it reaches, repeated across four open models spanning 4 to 14 billion parameters, which separates a limitation of one model from a limitation of the approach.

The conclusions are based on controlled experiments with real indoor radar and IR data, repeated across multiple seeds.

The outcome is negative, and is reported as such. In the main hyperparameter-search comparison, the LLM optimizer did not improve on the conventional methods: it was beaten by random search and by a deterministic rule-based controller. Last statement is bounded by the comparison it describes. A later diagnostic experiment, which adds a per-epoch training-curve summary to the payload, improves the rule-based controller result. The value of the work therefore lies less in the ranking itself than in what the controlled comparison isolates, that the information each method receives, rather than the strategy that consumes it, is where the limitation lies.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 reviews the background needed to follow the work: indoor localization, LSTM networks, conventional hyperparameter optimization methods, and the emerging use of LLMs as optimizers, closing with a discussion of related work. Chapter 3 presents the main contribution, describing the agent LLM and the conventional methods in detail. Chapter 4 reports the experimental evaluation: it details the methodology, presents the numerical comparison, and through a set of evaluation experiments that show where LLMs succeed and where failed, is the per-epoch training curve better than a search strategy. Chapter 5 turns to the thesis's motivating question, freezing the architecture and letting the LLM reshape the training loss from motion descriptors, and asks whether the model's knowledge of human movement adds anything beyond the extra tunable parameters. Chapter 6 summarizes the findings and outlines directions for future work.

Chapter 2

Background and Related Work

This chapter provides the background needed to follow the rest of the thesis. It introduces indoor localization and the role of radar (Section 2.1), the hyperparameter optimization problem and the conventional methods used as baselines (Section 2.2), and the recent idea of using large language models as optimizers (Section 2.3). It closes with a review of related work (Section 2.5).

2.1 Indoor Localization

Localization is the task of estimating the position of a target within a known baseline frame. Outdoors, this is solved by global navigation satellite systems, but their signals are attenuated and reflected inside buildings, which makes them unreliable for indoor use [18]. A large body of work has therefore studied indoor localization using other technologies, including Wi-Fi and Bluetooth signal-strength fingerprinting, ultra-wideband ranging, inertial sensors, cameras, and radar [18]. An early and influential example is the RADAR system, which located users indoors from Wi-Fi signal-strength measurements [2].

These technologies differ along several practical axes: cost, whether the target must carry a device, robustness to lighting and occlusion, and how much they intrude on privacy. Camera-based systems can be accurate but raise clear privacy concerns and depend on good lighting. Device-based methods require the person to carry a tag or smartphone, which is not always acceptable. Device-free environmental sensing, in which fixed sensors observe the space without any cooperation from the target, is attractive for ambient applications such as assisted living and smart buildings.

Radar is a strong candidate for device-free indoor sensing. It works in darkness, is robust to lighting, and does not record a recognizable image of the person, which preserves privacy. A low-cost radar reports, for each frame, compact measurements such as the angle and range of the dominant reflection together with its magnitude. Converting this stream of measurements into a position on the floor plan can be approached as a regression problem, in which a model maps a window of recent measurements to coordinates. Classical approaches rely on explicit geometric or statistical models, whereas learning-based approaches fit the mapping directly from data; the latter are well suited to the noisy, multipath-affected signals seen indoors. Deep learning has been applied widely

to this setting, including human activity recognition and tracking from millimeter-wave radar point clouds [13], and machine-learning methods for indoor positioning are now surveyed as a field in their own right [18].

2.2 Hyperparameter Optimization

The parameters of a neural network(weights) are learned from data during training. A second group of settings, the hyperparameters, must be fixed before training starts and are not adjusted by gradient descent. For the localizer, these include architectural choices, such as the number of LSTM layers and hidden units, and training choices, such as the learning rate, batch size, dropout rate, weight decay, and the length of the input window. The quality of the final model depends strongly on these values.

Hyperparameter optimization (HPO) is the problem of choosing the hyperparameters that yield the best model according to a chosen criterion, typically the error on a validation set. It is a black-box optimization problem: the relationship between a configuration and its resulting performance has no simple analytic form, and each evaluation is expensive, because it requires training a model. The goal is therefore not only to find a good configuration, but to find it using as few evaluations as possible.

2.2.1 Grid and Random Search

The most direct HPO methods are grid search and random search. Grid search evaluates every combination of a discrete set of values for each hyperparameter. It is simple but scales poorly: the number of combinations grows exponentially with the number of hyperparameters, a difficulty known as the curse of dimensionality. Random search instead samples configurations at random from the search space. Despite its simplicity, random search is often more efficient than grid search, because in typical problems only a few hyperparameters matter, and random sampling explores those important dimensions more thoroughly for a given budget [5].

2.2.2 Bayesian Optimization and Optuna

Bayesian optimization is a more sample-efficient family of methods. It builds a probabilistic surrogate model of the relationship between configurations and performance, and uses an acquisition function to decide which configuration to evaluate next, balancing the exploration of uncertain regions against the exploitation of promising ones [12]. A widely used variant replaces a Gaussian-process surrogate with the Tree-structured Parzen Estimator (TPE), which scales better to mixed and conditional search spaces [4]. These methods form part of the broader field of automated machine learning, which seeks to reduce the manual effort of model selection and tuning.

Optuna is a popular open-source framework that implements these ideas [1]. It provides a TPE-based sampler, support for defining search spaces programmatically, and pruning mechanisms that stop unpromising trials early. In this thesis, Optuna serves as the strongest conventional baseline against which the LLM-driven optimizer is compared.

2.2.3 Overtuning and Validation Overfitting

Every method above selects a configuration by minimising an estimate of generalization error computed on held-out data. That estimate is itself a random quantity, and searching hard enough against it is a form of fitting. Cawley and Talbot set this out for model selection generally: because the selection criterion has non-negligible variance, a search that drives it down aggressively can overfit the criterion, and the resulting model then generalizes worse than the score suggested [7]. They also note the second-order consequence, that performance reported on the same data used for selection is optimistically biased, which is why the protocol in this thesis selects on validation and reports on a test split that the search never sees.

Recent work has measured the effect specifically for hyperparameter optimization. Schneider et al. term it overtuning and, reanalysing a large body of HPO benchmark data, they report it to be more common than usually assumed: typically mild, occasionally severe, and in roughly one case in ten severe enough that the configuration chosen as optimal generalizes worse than the default it was meant to improve [11].

This matters for the present study in two ways. It supplies a mechanism that is not specific to language models, so it must be considered before any divergence between validation and test score is attributed to the LLM; and the failure mode it describes (a searched configuration losing to an untuned default) is the shape of the result reported in Chapter 4. Section 4.3.6 returns to it with the measurements from this experiment.

2.3 Large Language Models as Optimizers

Large language models (LLMs) are neural networks, almost always based on the Transformer architecture [15], trained on very large text corpora to predict and generate text [6, 14]. Beyond text generation, they have shown an ability to follow instructions and to reason over problems presented to them in natural language, including problems with numerical and structured content, particularly when prompted to reason in intermediate steps [16]. This has motivated researchers to ask whether an LLM can act as an optimizer.

The idea is to describe the optimization problem to the LLM in text, provide it with the history of configurations tried so far together with their measured scores, and ask it to propose the next configuration. The LLM thus plays the role that a sampler or a surrogate model plays in conventional HPO, but it reasons about the trajectory of past trials in language rather than through an explicit statistical model. Yang et al. demonstrated this approach, using an LLM to iteratively propose candidate solutions from a natural-language description of the problem and its previous attempts [17].

Using an LLM in this way has potential advantages and clear risks. On the one hand, it can draw on patterns and prior knowledge that a blind sampler cannot, and it may reach good configurations in fewer trials. On the other hand, it offers no convergence guarantees, may propose invalid or out-of-range values, and can be inconsistent across runs. These risks make it essential to constrain and validate the model's proposals. The optimizer studied in this thesis is therefore a single-agent design in which the LLM

proposes configurations from the history of past trials, and every proposal is checked against the search space before it is used.

This direction has grown quickly. Beyond the optimization-by-prompting approach of Yang et al. [17], LLMs have been used to drive neural architecture search [19] and to enhance the components of Bayesian optimization itself [10].

Closest to the present work is AgentHPO [9], which applies an LLM agent to hyperparameter optimization directly. There the model is given a description of the task and the dataset, runs experiments, and refines the configuration from the results of earlier trials. They report that across twelve machine-learning tasks the agent matches and often surpasses the best human trials, while producing an interpretable record of why each change was made. That is a positive result for the same basic idea this thesis tests, and the disagreement with the findings reported here is worth stating simply.

These studies establish that an LLM can steer a search loop; whether it does so better than the conventional methods, on a concrete engineering task, is the question this thesis takes up.

2.4 Domain Knowledge and Human Motion

Tuning hyperparameters, whichever method does it, treats the network as a black box and asks only how well a configuration scores. It makes no use of what is known about the problem. Here, that the target is a person walking through a room, whose movement obeys physical regularities.

A separate line of work argues that encoding such prior knowledge into the model or its training objective can improve data efficiency and generalization, especially when data is limited. Physics-informed and domain-informed learning, for instance, add terms to the loss that penalize physically implausible predictions rather than leaving the network to infer every constraint from data alone.

A walking person moves at bounded speed, changes velocity smoothly rather than instantaneously, dwells and resumes in characteristic stop-and-go patterns, and turns with limited angular rate. These properties can be summarized as interpretable motion descriptors (speed and its distribution, acceleration, turn rate, path sinuosity, and the frequency of stops) computed from the trajectory. Such descriptors are the everyday vocabulary of human-motion analysis and appear throughout radar-based activity recognition, where models are built specifically to exploit the temporal and kinematic structure of movement [13].

Two uses of this knowledge are relevant to this thesis. The first is diagnostic: breaking a model's error down by motion regime (slow, medium, and fast, or stationary versus moving) reveals where it fails, which a single aggregate error hides. The second is prescriptive: a training objective can be shaped to reflect motion, for example by weighting the loss toward the regimes a model handles worst, or by penalizing predicted trajectories whose implied speed or acceleration exceeds what a person can produce. This is the setting of Chapter 5, which asks whether a language model, given such motion descriptors in natural language, can propose useful changes to the training loss and whether its broad knowledge of how people move adds anything beyond the extra flexibility that a richer

objective provides.

2.5 Related Work

The work in this thesis sits at the intersection of three lines of research:

The first is learning-based indoor localization, which applies machine learning, including deep and recurrent networks, to estimate position from sensor streams such as radar or radio signals [2, 13]. These works share with this project the localization task and the use of sequence models, but they focus on the localization model itself rather than on how its hyperparameters are chosen. Those sequence models are for the most part built on the Long Short-Term Memory cell, and its mechanism is worth stating, because it is the architecture this thesis tunes. A recurrent network carries a hidden state forward as $h_t = \tanh(Wx_t + Uh_{t-1} + b)$, so the gradient arriving at an early step is a product of one factor for every step in between. Bengio et al. showed that the resulting vanishing and exploding gradients are not a defect of a particular optimizer but a structural tension: a recurrence that holds information stably across many steps is, for that very reason, one whose gradients decay across them [3]. Hochreiter and Schmidhuber's LSTM answers it by carrying a separate cell state that is updated additively rather than by repeated multiplication [8],

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t, \qquad h_t = o_t \odot \tanh(c_t), \tag{2.1}$$

where g_t is a candidate update and the gates f_t, i_t and o_t are themselves learned functions of x_t and h_{t-1}, squashed into (0,1) by a logistic sigmoid, which decide respectively what is erased from the cell, what is written into it, and what is exposed as output. Because the path from c_{t-1} to c_t is a gated sum rather than a matrix product, a forget gate held near one lets information and gradient cross many steps without being rescaled at each one, and that is what makes the cell trainable on sequences that defeat the plain recurrence.

The mechanism also fixes what there is to tune, which is where this thesis enters. The width of h_t and c_t sets the capacity of the summary a window is compressed into, cells may be stacked so that one layer's output sequence becomes the next one's input, and the window length decides how much history a single prediction may draw on. Those are three of the nine hyperparameters searched here (Table 3.1), and the localizer that instantiates them is described in Section 4.1.2.

The second is hyperparameter optimization, including random search, Bayesian optimization, and rule-based methods used here as baselines [1,5,12]. This literature provides the methods that the LLM optimizer is compared against, but it does not consider an LLM as the search strategy.

The third is the emerging use of LLMs as optimizers and for automated machine learning [10, 17, 19]. These works establish that an LLM can drive an optimization loop, but they typically target generic benchmarks rather than the tuning of a compact model under the accuracy, sample-efficiency, and robustness constraints of an edge-deployed localizer.

To the best of the author's knowledge, few works bring these three strands together by using a single-agent LLM to optimize the hyperparameters of a small LSTM localizer

and benchmarking it, under an identical protocol, against the conventional methods. Addressing this gap is the aim of the present thesis.

Chapter 3

LLM-Driven Hyperparameter Optimization

This chapter presents the main contribution of this work: an agent LLM optimizer for the small LSTM localizer and conventional methods for tuning hyperparameters. Section 3.1 describes the overall approach and the principle of a comparison. Section 3.2 defines the shared search space and the way each candidate configuration is scored. Section 3.3 details the agent, and Section 3.4 the strict validation that constrains its proposals. Section 3.5 describes the four conventional methods. Section 3.6 outlines the motion-aware extension of the training objective that Chapter 5 takes up.

3.1 Overview of the Approach

The aim is to observe LLM behavior, improvements, degradations, and where it helps. To do so, every method optimizes the same LSTM model, discrete space of hyperparameter values, training and scoring protocol. The difference between methods is how the metric is used to choose the next configuration. Any difference in the final result can be attributed to the search strategy for conventional methods.

Within this common frame, five methods are compared: a fixed baseline, random search, a rule-based search, Optuna, and the agent. Figure 3.1 shows the loop this frame defines, which is explained in detail in chapter 4.

3.2 Search Space and Scoring

The search space is a fixed grid of discrete values for nine hyperparameters, listed in Table 3.1. It mixes architectural choices, which change the network and therefore require it to be rebuilt. This grid is the single source of truth for every method: the random, Optuna, rule-based, and LLM arms all draw from these identical value sets, so they explore the same territory. The fixed baseline trains at the window length dictated by the dataset. To keep the comparison fair, that window length is also made reachable inside the shared grid, so that any search method could in principle rediscover the baseline configuration.

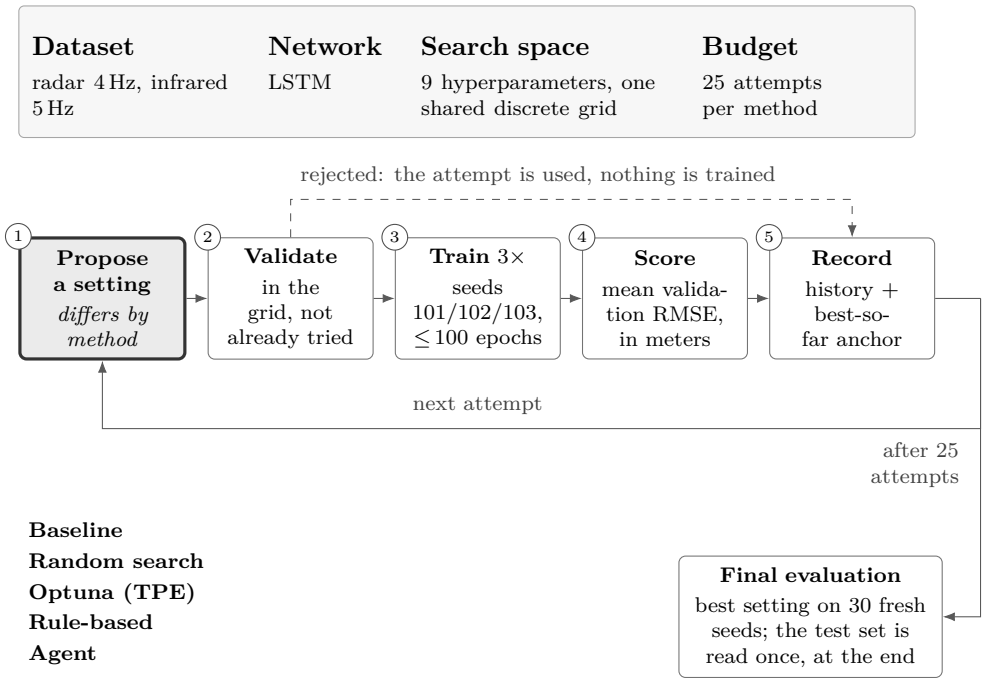


Figure 3.1. The optimization cycle shared by the five methods. Everything outside step 1 is fixed protocol: the same dataset, network, nine-hyperparameter grid, budget of 25 attempts, and the same score, the mean validation RMSE in meters over three trainings. Only the proposal made at step 1 differs between the arms. Step 2 applies to the rule-based and agent arms alone. Random search and Optuna draw from the grid by construction, so their proposals are always valid, though they may repeat a setting already evaluated. A rejected proposal still consumes one of the 25 attempts and is recorded without being trained.

Table 3.1. Shared hyperparameter search space.

Hyperparameter	**Allowed values**
Learning rate	$10^{-5},\ 3\times10^{-5},\ 10^{-4},\ 3\times10^{-4},\ 10^{-3},\ 3\times10^{-3}$
Weight decay	$0,\ 10^{-6},\ 10^{-5},\ 10^{-4},\ 10^{-3}$
Dropout	0.0 to 0.5 in steps of 0.05
Batch size	32, 64, 128
LSTM hidden size	64, 128, 256
LSTM layers	1, 2, 3
Window size	10, 12, 20, 30, 40, 50
Optimizer	Adam, AdamW
Early-stopping patience	8, 12, 16

Each candidate configuration is scored in the same way for every method. The configuration is trained from scratch three times, once for each of three fixed random seeds,

with a maximum of one hundred epochs and early stopping on the validation set. Its score is the mean validation root-mean-square error (RMSE) of the position prediction, measured in meters at the early-stopping best epoch and averaged across the three seeds; lower is better. The same RMSE in meters is later used as the headline test metric, so the configuration selected during the search and the figure reported in the final evaluation are expressed in identical units. (Chapter 4).

3.3 Agent

The proposed optimizer uses a single large language model as the search strategy. The model is served locally and is queried once per round.

The optimizer runs for 25 rounds. In each round, the model is given a textual description of the situation and asked to propose the next configuration to try. The idea is to prevent the raw data/numbers from reaching the LLM, so here we use qualitative labels to make it understandable to the LLM. The input contains: prompt, best configuration so far, anchor, summary of the recent attempts, already tried, and summary of observed patterns. Moreover, the history is presented as a compact, qualitative summary rather than raw tables of numbers, so that the model reasons about the trend of the search rather than about individual figures.

The model is instructed to propose a change relative to the anchor, restricted to values from the allowed grid, and not to repeat a configuration that has already been tried. It must reply in a plain text format that states a diagnosis of the current situation (for example, healthy, possible overfitting, plateau), a short strategy, the concrete changes proposed, a one-sentence reason, and a confidence level. This structured reply is parsed into a candidate configuration. The proposed change is then applied to the anchor to form the configuration that will be evaluated.

This design places the LLM in the slot that every search method fills: the component that chooses which configuration to evaluate next. The methods differ in what that choice is based on. The full prompts, a real input and a real reply are reproduced in Appendix C.

3.4 Strict Validation of Proposals

A large language model can produce malformed outputs, values outside the allowed grid, parameters that are not part of the search space, architectural changes when those are not allowed, or a repeat of a configuration already tried. If outputs were silently corrected, the LLM would be given help that the other methods do not receive, and the comparison would no longer be fair. To prevent this, every proposal passes through a strict validation step before it is used.

The validator checks each proposed change against the shared grid and the set of already-tried configurations. A proposal is rejected if it baselines an unknown parameter, proposes an architectural change while architecture is frozen, sets a value that is not in the grid, or reproduces a configuration already evaluated. In the main configuration of the experiments, no silent repair is performed: a rejected proposal is never quietly clamped or corrected into a valid one. Instead, a short explanation of the problem is

fed back to the model, which is asked to try again once, if the second attempt also fails, the round is recorded as a rejected proposal and no model is trained. A configuration is accepted only if it is valid, new, and already in the shared grid. Accepted values are stored in their canonical grid form (for example, an integer batch size), which changes their representation but never their value.

3.5 Conventional Methods

3.5.1 Fixed Baseline

The fixed baseline trains a single configuration and evaluates it, performing no search at all. It is the baseline point against which every search method is measured, so what it represents matters. This is manual tuning carried out before this study began. Its values are listed in Table 3.2.

Table 3.2. The fixed baseline configuration, reached by manual tuning. Each value lies inside the shared search space of Table 3.1, so any search method could in principle rediscover it. The window length of 12 samples is the length specified for the dataset, and is injected into the shared grid for that reason. parameters.

Hyperparameter	**Value**
Learning rate	10^{-3}
Weight decay	10^{-3}
Dropout	0.30
Batch size	64
LSTM hidden units	64
LSTM layers	2
Window size	12
Optimizer	Adam
Early-stopping patience	12

The configuration was reached in fifteen trials, each a single deliberate change from the one before. Selection was based on validation error and the shape of the convergence curve. Among settings of comparable error, the one whose training and validation curves were tighter and more stable was kept. The test set was never inspected at any point during tuning, so the baseline carries no test-set advantage. This is a statement about how the reference was obtained, not about the block itself: over the thesis as a whole the radar test set was evaluated repeatedly, and Section 4.1.1 sets out what it therefore is and is not. Tuning stopped at the fifteenth trial because the object of this study is whether an agent LLM can improve on a human-tuned starting point, not how far manual tuning can itself be pushed. The full trajectory, every change made at each step, the observation that motivated it, and the resulting error is given in Appendix A.

The split protocol was not fixed across the manual search: the first ten trials used contiguous 70/15/15 and 80/10/10 splits, and only the last five used the non-contiguous protocol split defined in Section 3.2.

3.5.2 Random Search

Random search draws a configuration by sampling each hyperparameter uniformly from the shared grid, evaluates it with the common scoring procedure, and repeats this for the same number of rounds as the other methods. It keeps the configuration with the lowest mean validation RMSE. Its sampling uses a separate random stream, so that the choice of configurations does not interfere with the fixed seeds used for training. Despite its simplicity, random search is a strong and widely used baseline [5].

3.5.3 Rule-Based Search

Rule-based search is a deterministic controller that uses the same diagnostic categories as the LLM optimizer, such as healthy, possible overfitting, possible underfitting, plateau, unstable, and inconclusive, which are derived from the same qualitative reading of training and validation behavior. The difference is how those diagnoses are acted upon: instead of natural-language reasoning, the controller walks a fixed priority list of responses. For example, on a plateau it lowers the learning rate by one step on the grid, on underfitting it raises it, on overfitting it increases dropout. It changes a single hyperparameter per round, stepping one position along the shared grid relative to the anchor, and it selects the first move that is feasible and has not already been tried. When architectural changes are permitted, these moves may also alter the number of LSTM units or layers, which rebuilds the network.The controller is given in full, as pseudocode and as the move tables it walks, in Appendix D.

3.5.4 Optuna (Bayesian Optimization)

A framework that reduces the manual effort and tuning. It explores the same discrete grid as the other methods, but its sampler is based on the Tree-structured Parzen Estimator [4], a form of Bayesian optimization that uses the scores of past trials to bias later trials toward promising regions of the space [12]. The sampler is seeded at 1000 and configured with five start-up trials, so the first five attempts are drawn at random and the remaining twenty are guided by the displacements. Two choices keep the arm comparable with the others. First, values are drawn from the shared grid as categorical choices rather than from continuous ranges, so the reachable set is the Table 3.1 and the comparison with random search isolates the sampler alone. The cost is that the surrogate treats ordered numeric values as unordered labels: two adjacent learning rates are as unrelated to it as Adam is to AdamW, so the ordering that a continuous changes would exploit is not available to it. Second, Optuna's pruning mechanism, which halts unpromising trials early, is not used here: every setting is trained to completion under the common scoring procedure, so an attempt costs the same in this arm as in any other.

3.6 Extending the Training Objective

The agent described so far reasons about the search itself. It sees which settings have been tried, how each behaved, qualitative labels, and proposes the next one to try. Nothing in

that loop calls on what a language model might know about the problem being solved: whatever it knows about how people move through a building is of no use in choosing between two learning rates.

The remainder of the thesis changes what the model is asked to reason about. The nine conventional hyperparameters are frozen at the baseline configuration, and a set of loss-shaping levers takes their place as the search space: weights on the slow, medium and fast motion regimes, together with penalties on implausible predicted velocity and on sudden changes in it. Alongside the search history, the model is shown how the tracked person actually moves, the distribution of speeds, the stop-and-go structure of the trajectory, and where each setting's error falls across those regimes. It can then act on that reading: predictions that degrade during stop-and-go movement, or error concentrated in the fast regime, identify the segments the localizer handles worst, which the model can answer by weighting those segments more heavily in the training objective.

An improvement alone would not settle the question, because a six-lever objective is more flexible than a plain mean-squared error; however its levers are set, so a gain might come from the added flexibility rather than from any understanding of movement. The comparison therefore holds the six levers constant across the searching arms. This is the subject of Chapter 5.

Chapter 4

Experimental Evaluation

4.1 Methodology

4.1.1 Dataset and Preprocessing

The experiments use an indoor radar recording (the `RadarEXP1` dataset). 2000 radar samples at 4 Hz; each frame is reduced to a few scalar measurements: the angle and range of the dominant reflection and its signal strength, and the regression target is the planar position (X, Y) of the tracked person, expressed in meters. The model reads a sliding window of 3 seconds, i.e. 12 consecutive frames, and predicts the position one frame after the end of that window.

Samples are divided into three sets before the sliding windows are formed, and each set is then windowed on its own to prevent data leakage. The order matters: had the windows been cut first and the resulting sequences divided afterwards, a window straddling a boundary would carry frames from one set into another, and a model evaluated on the test would have been trained on part of it. Both the input features and the target coordinates are standardized. Writing μ and σ for the mean and standard deviation of a quantity measured over the training data, every value x is replaced by

$$z = \frac{x - \mu}{\sigma}. \tag{4.1}$$

Two independent scalers are used. The first is fitted to the input features, with μ and σ computed per feature and pooled over all time steps inside the training windows, so that a given feature is scaled identically wherever it occurs in a window. The second is fitted to the targets, giving the X and Y coordinates each their own mean and standard deviation,

The train, validation, and test partitions are split and fixed for all experiments, so methods do nothing with this part. Frames 0–1200 form the training set, frames 1200–1600 the test set, and frames 1600–2000 the validation set. Figure 4.1 shows the layout, and Figure 4.2 the ground-truth path in each block.

Within any single optimization experiment the protocol is: a method ranks candidates on validation RMSE alone, and the test block is read once, after the selection is fixed, for the final evaluation. The radar test set is one fixed stretch of one recording,

and it was evaluated repeatedly over the development with its results. It is therefore a fixed held-out test block, not an untouched external test set, and choices made during development were not made blind to it. Nothing about the protocol is invalidated by this, because every method was measured against the same block under the same rule, so the comparisons between arms remain fair. What it rules out is reading the absolute test figures as an unbiased estimate of performance on new data. Establishing that would require independent trajectories, subjects, rooms or recordings, which Section 6.2 takes up. The same holds of the infrared blocks of Chapter 5, which were used less but under the same conditions.

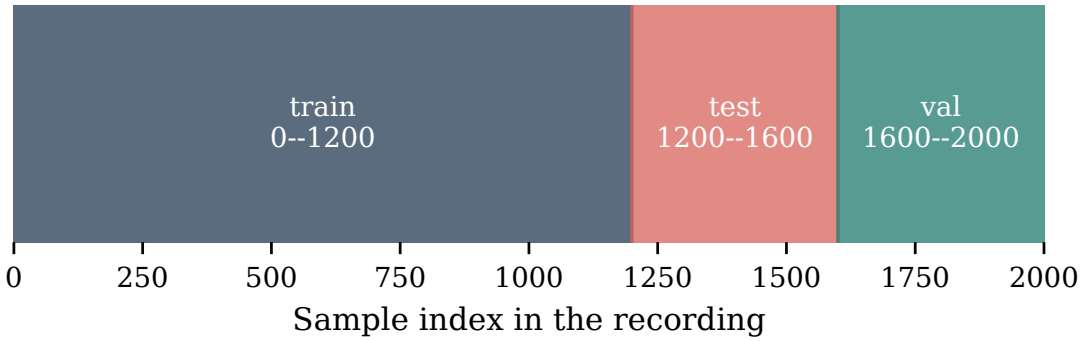


Figure 4.1. The dataset split of `RadarEXP1` recording.

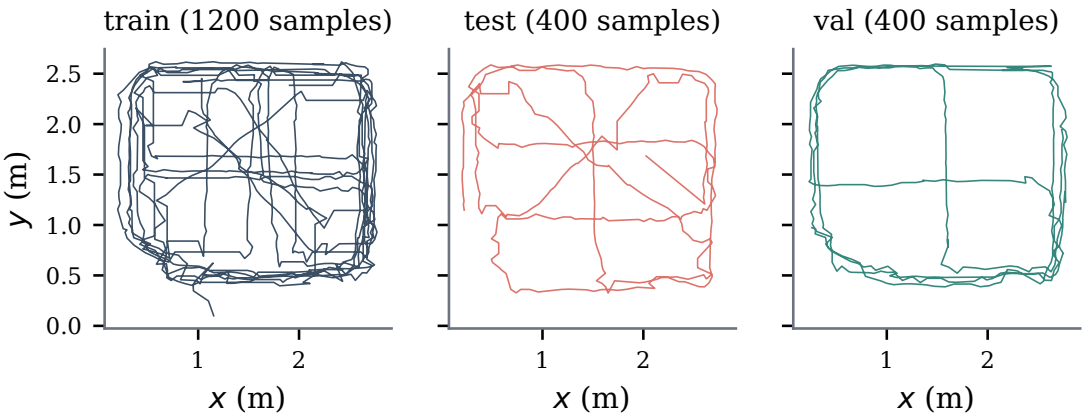


Figure 4.2. Ground-truth trajectory in each partition.

4.1.2 Localizer and Training

The Neural Network model is a Long Short-Term Memory (LSTM), shown in Figure 4.3. A sliding window of consecutive sensor frames, for the radar dataset, 12 frames of 3

features each, is read by a stacked LSTM. Only the hidden state after the last frame is kept; it passes through a fully connected head, a hidden layer of 64 units with a ReLU activation and dropout, and a final linear layer that outputs the two coordinates $(\hat{x}, \hat{y})$. The predicted position is the one held at the frame immediately following the window, so the network never reads a frame at or after the instant it is asked about: the input is strictly past, and the target is one step into its future.

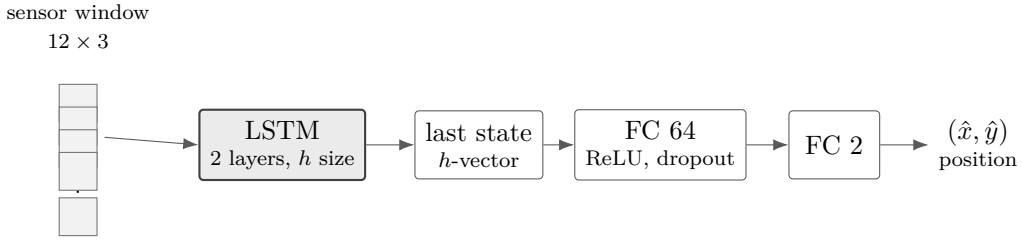


Figure 4.3. The LSTM architecture.

The design is a sequence-to-one regressor rather than a tracker. Each window is processed independently. The recurrent state is initialised to zero at the start of every window and discarded once the window has been read, so the network carries no memory from one prediction to the next, and successive outputs are related only through the overlap of their input windows. The LSTM therefore acts as an encoder that compresses a window into a fixed-length summary, and the fully connected head maps that summary to a coordinate pair.

Written in terms of tensor shapes, with B the batch size, T the window length, F the number of input features and h the hidden size, one forward pass is

$$\begin{aligned}(B,T,F) \xrightarrow{\text{LSTM}} (B,T,h) \xrightarrow{\text{last step}} (B,h) \\ \xrightarrow{\text{FC, ReLU, dropout}} (B,64) \xrightarrow{\text{FC}} (B,2).\end{aligned} \tag{4.2}$$

The number of hidden size h and the number of LSTM layers are part of the search space (Table 3.1). The full class is listed in Appendix B.

Training minimizes the mean-squared error between predicted and true coordinates with the Adam or AdamW optimizer (a searchable choice), using dropout and weight decay as regularization. Early stopping monitors the validation RMSE in meters and restores the weights of the best epoch. The patience (8, 12, or 16) is itself a searchable hyperparameter, and each training is capped at 100 epochs.

4.1.3 Search Protocol and Scoring

Every method is given the same budget of 25 attempts and draws from the same discrete grid (Table 3.1). Each candidate configuration is scored by training it three times from scratch on the fixed inner seeds {101,102,103} and taking the mean validation RMSE (in meters, at the early-stopping best epoch). Then final evaluation uses 30 seeds. The configuration with the lowest score is selected. The test set is considered only after a setup has selected its best configuration.

4.1.4 Methods Compared and Implementation

Five methods are compared under this identical protocol: the fixed baseline, random search, the rule-based controller, Optuna (TPE sampler, seed 1000, 5 start-up trials), and the agent LLM. The LLM arm is served locally through Ollama and reported for four open models spanning 4 to 14 billion parameters(`llama3:8b`, `phi4:14b`, `nemotron-3-nano:4b` and `gemma4-12B`), each queried once per round. Reporting several models separates a limitation of one particular model from a limitation of the approach.

The pipeline is implemented in Python 3.13. The LSTM is built and trained with PyTorch, the standardization of Section 4.1.1 and the coefficient of determination reported in Section 4.2.1 use scikit-learn's `StandardScaler` and `r2_score`. The Optuna arm uses that library's TPE sampler [1], and the language models are accessed through the Ollama client of the `autogen-ext` package. Training was carried out on a laptop first, then, to accelerate the work use the University server.

Only one decoding parameter is set for the language models: the temperature is fixed at 0.2 for every model and every call. The context length and the maximum number of generated tokens are left at each model's own default. Each call is given a 300-second timeout. A call that exceeds it is recorded as a failed attempt and retried once; if the retry also fails, the attempt is consumed without any configuration being trained. This is the mechanism behind the timeout entries in Table 4.3, and it is why a lost attempt is not by itself evidence about the quality of a model's proposals.

4.1.5 Evaluation and Metrics

To measure robustness, the experiment has been done on 10 complete optimization runs per LLM (4 LLMs), hence 40 full searches and 30 seeds for final evaluation of each selected configuration (17, 42, 73, 128, 256, 314, 451, 512, 666, 777, 888, 1024, 1234, 1492, 1618, 1729, 2003, 2048, 2718, 3141, 4096, 4242, 5000, 5678, 6174, 7777, 8191, 8675, 9001, 9999).

Each complete optimization run has its own run seed, which controls the stochastic parts of the search, including random-search sampling. The three fixed training seeds are used only when scoring candidate configurations, while the 30 fresh evaluation seeds are used only for the final evaluation.

Every quantity reported in this chapter is computed after inverting the target scaling, so the errors are expressed in meters. Write N for the number of evaluated windows, $\mathbf{y}_i$ for the true position of window i, $\hat{\mathbf{y}}_i$ for the prediction, and $d_i = \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|_2$ for the Euclidean distance between them. The headline error is the root-mean-square error taken over both coordinates,

$$\mathrm{RMSE} \;=\; \sqrt{\frac{1}{2N}\sum_{i=1}^{N}\left\|\hat{\mathbf{y}}_i - \mathbf{y}_i\right\|_2^2} \qquad [\mathrm{m}]. \tag{4.3}$$

The coefficient of determination is computed separately for each coordinate and averaged with equal weight,

$$R^2 \;=\; \frac{1}{2}\sum_{c\in\{x,\,y\}}\left(1 - \frac{\sum_i (\hat{y}_{ic} - y_{ic})^2}{\sum_i (y_{ic} - \bar{y}_c)^2}\right), \tag{4.4}$$

where $\bar{y}_c$ is the mean of coordinate c over the evaluated windows. It is dimensionless and is reported alongside the RMSE.

Two further quantities are tracked once per epoch while a network trains: the value of the training objective and the mean absolute error $\frac{1}{2N}\sum_i \sum_c |\hat{y}_{ic} - y_{ic}|$. Both are evaluated on standardized data, and they serve to monitor convergence and the training curves. Table 4.1 collects these quantities.

Table 4.1. Quantities computed during and after training. Those evaluated in standardized space are dimensionless and are used only to monitor a single training run; every quantity used to compare methods is expressed in meters.

Quantity	Space	Units	Role
Training objective	standardized	—	per-epoch monitoring
Mean absolute error	standardized	—	per-epoch monitoring
Position MSE	meters	m^2	early stopping; its root is the search score
RMSE, Eq. (4.3)	meters	m	headline test metric
Distances d_i	meters	m	mean, median, maximum, standard deviation
R^2, Eq. (4.4)	—	—	reported with the RMSE

The reported test figure for one search is the mean over those 30 evaluations, so the selection and the measurement never share randomness. Results are then aggregated across searches and reported along the four axes introduced in Chapter 1.

4.2 Numerical Results

4.2.1 Final Accuracy

Table 4.2 reports the outcome of all five methods, and average result of four LLM configurations. The result is negative: under payload, no search method achieved a lower mean test RMSE than the fixed baseline in this comparison. Section 4.3 shows that the qualifier matters, since once the methods are given per-epoch curve information, one of them does beat the baseline. The baseline configuration attains 0.2308 m test RMSE. The best search method is rule-based controller, reaches 0.2371 m, which is 2.7% higher. Random search follows at 0.2413 m (+4.6%), the LLM at 0.2455 m (+6.4%), and Optuna is last at 0.2614 m (+13.3%). The ordering by R^2 is identical.

Figure 4.4 shows the same result per search. Each point is one complete search, the horizontal marker is the mean, and the whiskers are one standard deviation. The dashed line marks the baseline. Random search is the only method that lands below the baseline. That success is worth examining, since it is the only selection in this chapter to beat the baseline. On search seed 42, random search chooses a network of 128 hidden units in a single LSTM layer, trained with weight decay 10^{-5} and no dropout, which reaches 0.2197 m against the baseline's 0.2308 m.

Table 4.2. Search-method comparison on the radar dataset, pooled over the four LLM configurations and 30 evaluation seeds ($n = 40$ full optimization runs). Test RMSE and R^2 are means over seeds per search. Win rate is the fraction of searches whose selection beats the fixed baseline. AUC is the normalised area under the best-so-far validation trajectory (lower is better). Standard deviations of zero reflect deterministic methods, not missing variation.

Method	RMSE (m)	R^2	vs. ref. (%)	Win rate	AUC
Fixed baseline	0.2308 ± 0.0000	0.907	+0.0	−−	−−
Rule-based	0.2371 ± 0.0000	0.903	+2.7	0.00	0.691
Random search	0.2413 ± 0.0099	0.898	+4.6	0.10	0.689
LLM	0.2455 ± 0.0058	0.895	+6.4	0.00	0.688
Optuna (TPE)	0.2614 ± 0.0000	0.884	+13.3	0.00	0.670

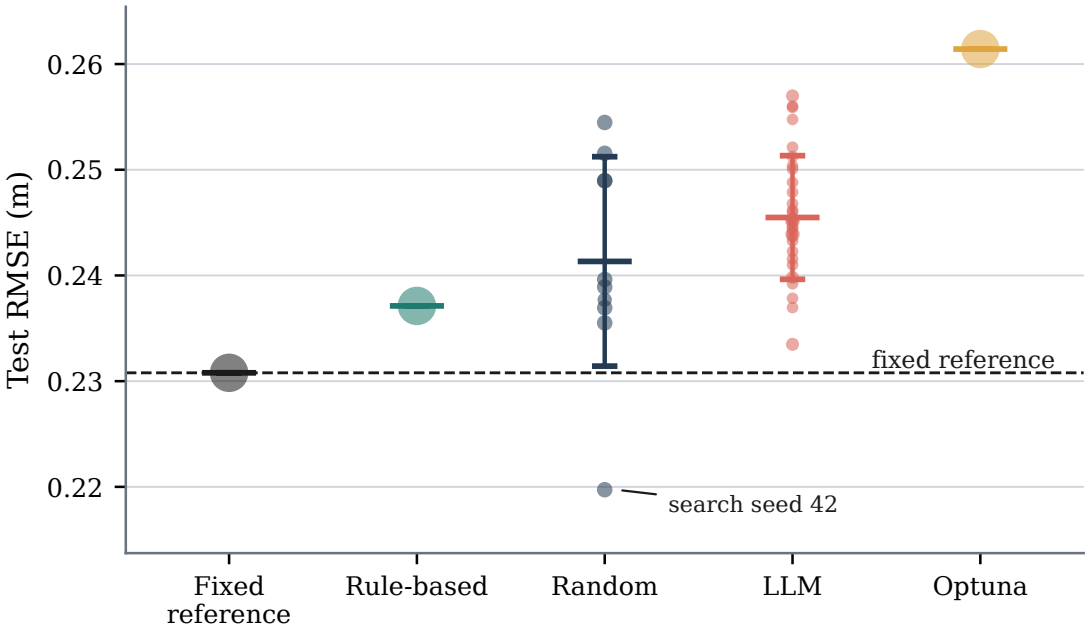


Figure 4.4. Test RMSE per method.

The gap in Table 4.2 is therefore not evidence that search is useless, because of the lack of information they use. We will see after adding the curve-aware to rule-based and agent it helps.

Paired per-seed differences confirm the ranking is not artificial. Against the LLM, the fixed baseline is better by 0.0147 m (95% CI $[0.0129, 0.0165]$, LLM better on 0 of 40 searches), the rule-based controller by 0.0084 m ($[0.0065, 0.0102]$, LLM better on 3), and random search by 0.0042 m ($[0.0006, 0.0078]$, LLM better on 15). The single comparison the LLM wins is against Optuna, by 0.0159 m ($[-0.0178, -0.0141]$), on every one of the

40 searches(10 full optimization runs for each LLM model). Optuna's weak showing is expected at this budget with 5 start-up trials, and a TPE sampler spends a fifth of 25 attempts sampling at random before its setup.

4.2.2 Reliability

Within base experiment only random search beats baseline reliably. On seed 42, the selection described in Section 4.2.1. Random search and the LLM vary across seeds, and their standard deviations differ slightly: 0.0099 m for random against 0.0058 m for the LLM.

4.2.3 How the Budget Is Spent

The accuracy figures of Section 4.2.1 understate the difference between the arms, because they do not spend the same budget. Every conventional method converts all 25 of its attempts into trained configurations: random search, the rule-based controller, and Optuna propose well-formed candidates by construction, so none of their attempts is wasted. The language model averages 16.6 trained attempts across all searches, leaving 8.4 that produce nothing. An attempt is lost when a proposal fails validation twice(by repeating a configuration already evaluated, or naming a value the grid search does not contain, or when no reply arrives within the timeout). Moreover, a proposal that survives validation is never a repeat; the number of attempts a model trains is also the number of distinct configurations it evaluates. How the losses divide between those causes differs between models.

Table 4.3. How each model spends its budget.

Model	Accepted	Lost	RMSE (m)
`nemotron-3-nano:4b`	24.2	0.8	0.2487 ± 0.0059
`gemma4-12B`	19.4	5.6	0.2437 ± 0.0038
`phi4:14b`	13.9	11.1	0.2440 ± 0.0085
`llama3:8b`	9.1	15.9	0.2453 ± 0.0028
Conventional methods	25.0	0.0	—

Table 4.3 and Figure 4.5 show how uneven this is. `llama3:8b` converts only 9.1 of 25 attempts into evaluated configurations, it spends nearly two thirds of its budget re-proposing settings it has already tried, while `nemotron-3-nano:4b` wastes almost nothing. The notable part is that these differences in effective budget barely move the final precision: all four models land between 0.2437 and 0.2487 m, and the model that explored most (`nemotron`) finished last. Whatever limits the LLM arm, it is not primarily the number of configurations it manages to try.

The same point appears in the coverage figures. Over all attempts, the LLM evaluates 16.6 distinct configurations on average against 25 for random search and the rule-based controller. Optuna evaluates 19, revisiting the same regions. Yet the LLM's accuracy sits

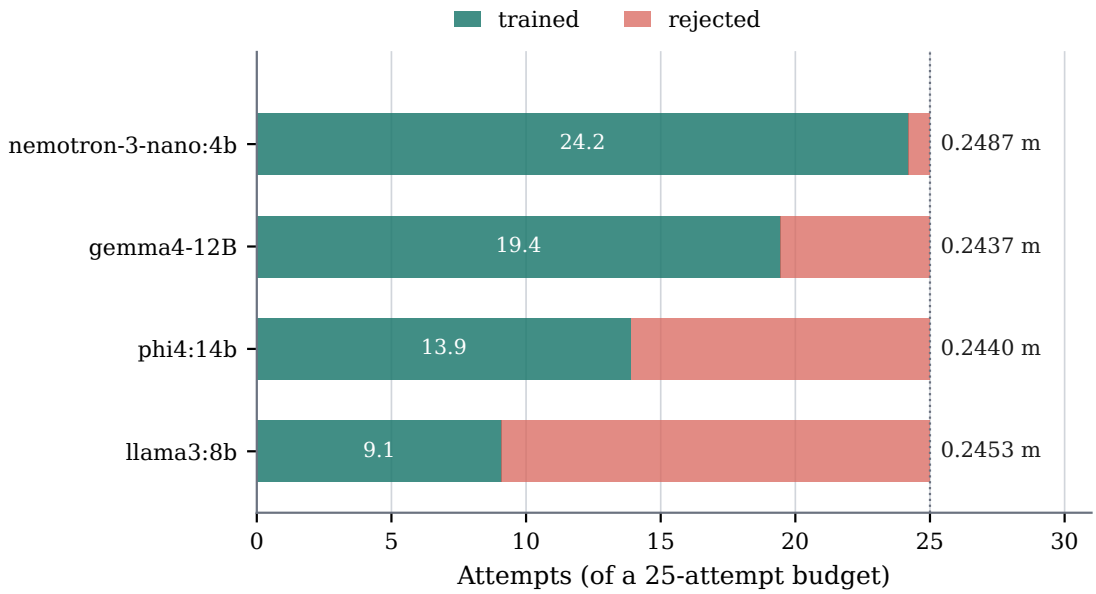


Figure 4.5. How each model spends the budget, with the resulting test RMSE on the right. The effective budget varies by a factor of 2.7 across models, yet the accuracy varies by less than 0.005 m, and the model that trains the most configurations, `nemotron-3-nano:4b`, is the least accurate.

between random and Optuna's rather than tracking coverage. This Chapter supports a narrower claim than the LLM explores little: It explores less and chooses no better among what it does explore.

The out-of-grid rejections repay a closer look, because two models arrive at them for different reasons. Of `llama3:8b`'s 17, ten propose a hidden width the grid does not contain, 512 on nine occasions and 384 on one, and are genuine attempts to leave the search space. Of `phi4:14b`'s 9, only two are values at all: the remaining seven write explanatory prose into the field where a value belongs, proposing, for instance, a learning rate of 0.001 is a moderate choice, balancing convergence speed and stability. Seven of `llama3:8b`'s are of a third kind, stating a transition rather than a value, as in `weight_decay=1e-05 -> 0.0001`; in three of those the intended destination was in fact a legal grid value.

The distinction matters for how the budget figures are read. A proposal that names an unavailable value is a search failure, and reflects on what the model understands about the search space it was given. A proposal whose value is unparsable, or which is legal but wrapped in commentary, is a formatting failure and reflects only on whether the model can hold to a five-line reply format. The validator cannot tell them apart, and Table 4.3 records both as lost attempts, exactly as it records `gemma4-12B`'s serving timeouts. Across the two models, fourteen of the twenty-six out-of-grid rejections are formatting rather than search failures, so the raw counts overstate how often these models genuinely tried to leave the grid.

4.2.4 Optimization Dynamics

The final column of Table 4.2 gives the normalised area under the best-so-far validation trajectory, where 1.0 would mean never improving on the baseline's validation score and lower values mean the score was driven down sooner. The four search methods are separated by less than 0.03: Optuna 0.670, LLM 0.688, random 0.689, rule-based 0.691. Optuna's advantage here alongside its worst final accuracy is not a contradiction, it drives the validation score down soonest and still generalises worst, which is the validation-to-test dissociation of Section 4.3.6 appearing in the dynamics column, and which the fresh-seed evaluation is designed to expose.

Median attempts-to-best tells the same story: Optuna reaches its eventual best at attempt 10, random at 14, the LLM at 17, and the rule-based controller at 20. No method plateaus early enough for the budget to be the binding constraint, and the two methods that improve fastest are also the two that generalise worst. Sample efficiency and final quality are, on this problem, close to unrelated.

4.3 Why the LLM Fails

The comparison so far makes it seem that the LLM does not help; here we explain why. This section reports three diagnostic experiments that vary what the proposer is shown while holding the history fixed. Each isolates one candidate explanation: insufficient information, unused history, and prompt-induced under-exploration.

4.3.1 Per-Epoch Curve Summaries

The fixed baseline was obtained by manual tuning, in which a human read the per-epoch validation curve. The arms see only an end-of-run aggregate. This experiment adds a compact per-epoch curve summary to the payload and repeats the optimization.

Figure 4.6 shows the outcome. Curve information helps both arms by a similar margin: the LLM(llama3:8b) improves from 0.2453 to 0.2356 m and the rule-based controller from 0.2371 to 0.2264 m. At 0.2264 m the curve-aware rule-based controller beats the fixed baseline. This supports the information hypothesis: what the earlier comparison lacked was the information (training curve) used by the method, not a better search strategy.

4.3.2 Does the LLM Use the History?

The LLM's advantage over other methods is that it reasons over history. This experiment tests that directly by manipulating the history while leaving the true search state intact: the shuffled condition permutes the trial-to-outcome mapping, and the empty condition removes the history entirely. If the history is being used, degrading it should degrade the result.

The opposite happens on the metric reports. Each condition was run as ten complete searches per model across five models, 145 searches in all, and Table 4.4 gives the six quantities with the interval on their means.

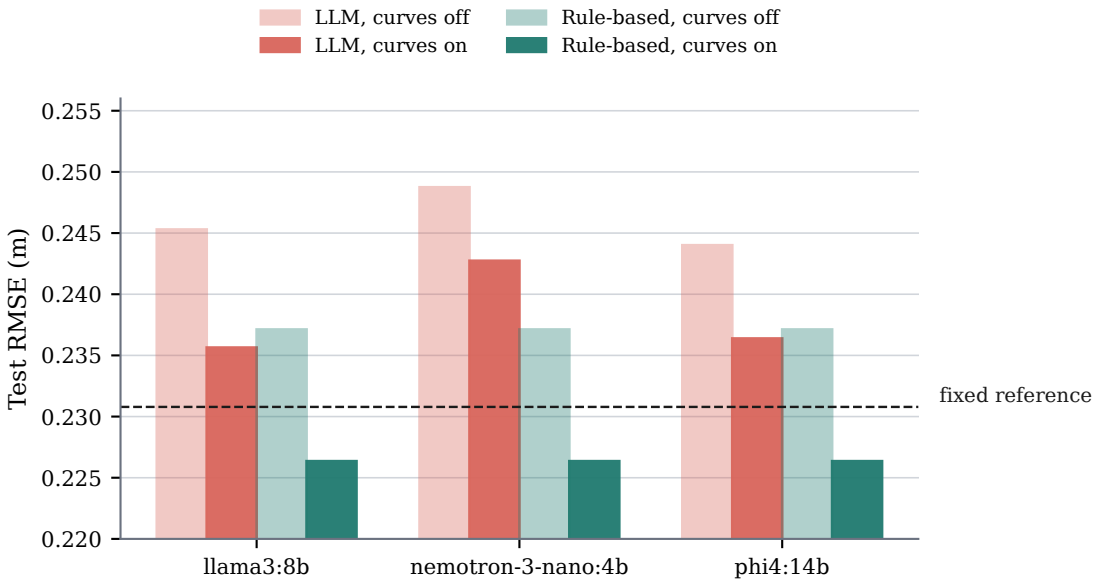


Figure 4.6. Effect of adding per-epoch training-curve summaries. Both methods improve, but the rule-based controller improves enough to cross below the fixed baseline, while the LLM given the same information does not.

Table 4.4. Each history condition over its complete searches, pooled across the five models (145 searches). Validation RMSE is the best score a search reached, the one its selection was made on; distinct counts the different configurations it evaluated; repeats is the share of the 25 attempts rejected for proposing a setting already tried; coverage is the mean fraction of each hyperparameter's allowed values the search visited; last improvement is the attempt at which its best score last moved.

History	Test RMSE (m)	Val. RMSE (m)	Distinct	Repeats	Coverage	Last impr.
real	0.2466 ± 0.0019	0.1973 ± 0.0015	18.2 ± 1.9	0.21 ± 0.07	0.56 ± 0.03	16.0 ± 2.0
shuffled	0.2428 ± 0.0016	0.2012 ± 0.0021	15.0 ± 2.0	0.30 ± 0.09	0.52 ± 0.03	14.0 ± 2.3
removed	0.2413 ± 0.0025	0.2058 ± 0.0036	10.8 ± 1.5	0.49 ± 0.07	0.44 ± 0.02	14.5 ± 2.2

Every column moves monotonically as the history is degraded, and the two outcome columns move in opposite directions. Test error falls from 0.2466 to 0.2413 m, intervals that barely overlap, while the validation error the search is actually minimizing rises from 0.1973 to 0.2058 m, intervals that do not overlap at all. The degraded conditions are therefore worse at the job they were set and better at the job they are judged on, which is the same dissociation Section 4.3.6 documents for the arms as a whole.

The behavioural columns rule out the explanation that first suggests itself. Removing the history cuts the distinct configurations evaluated from 18.2 to 10.8, more than doubles the share of attempts wasted on settings already tried, from 0.21 to 0.49, and lowers grid coverage from 0.56 to 0.44. A model with no record of what it has tried cannot avoid retrying it, so it searches a smaller region less efficiently and stops improving no later

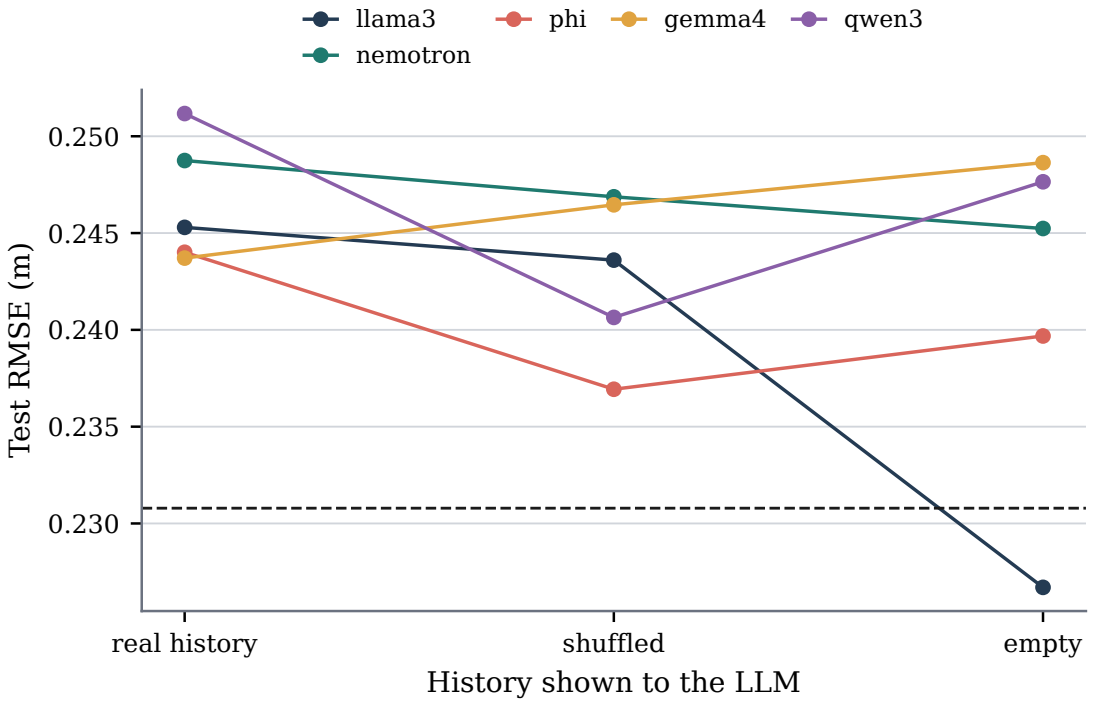


Figure 4.7. Test RMSE as the history shown to the LLM is degraded, one line per model. The pooled result improves as the history is degraded, but the models do not agree: `gemma4-12B` rises monotonically, `qwen3-14B` is best on a shuffled history, and only `llama3:8b` reaches the fixed baseline (dashed line), which it does with the history removed.

than before. The gain is not the product of a wider search; it is what remains when a search is limited, and the selection stays nearer the configuration the reference already represents.

Table 4.5. Each degraded condition paired against the same model's real-history searches, seed by seed. Negative means the degraded condition gave the lower test error; * marks a 95% interval excluding zero, and better on counts the shared seeds it won. The direction is not consistent across models: removing the history helps two of the five, and interrupts one, and settles nothing for the other two.

	shuffled − real		removed − real	
Model	Difference (m)	Better on	Difference (m)	Better on
`llama3:8b`	−0.0017	6/10	−0.0186*	10/10
`phi4:14b`	−0.0071*	7/10	−0.0043	7/10
`nemotron-3-nano:4b`	−0.0019	6/10	−0.0035*	6/10
`qwen3-14B`	−0.0105*	8/10	−0.0035	7/10
`gemma4-12B`	+0.0024	4/8	+0.0046*	1/8

Pairing within each model, in Table 4.5, shows that the pooled figure validates disagreement. Removing the history helps `llama3:8b` and `nemotron-3-nano:4b` by intervals excluding zero, leaves `phi4:14b` and `qwen3-14B` undecided, and degrades `gemma4-12B` by 0.0046 m, which loses on seven of its eight seeds. Shuffling is not harmless: it is the best condition for `qwen3-14B`, by 0.0105 m. No single statement about history use covers all five models, and any claim that the language model does not benefit from history has to be qualified by the one model that does.

The strongest case makes the mechanism clear. `llama3:8b` without a history returns the best language-model result in this experiment, 0.2267 m, which is also better than the fixed baseline. It reaches it by evaluating exactly three distinct configurations in every one of its ten searches, spending 88 per cent of its budget re-proposing settings it had already tried, and never improving after its opening attempts; its validation error, at 0.2282 m, is the worst of any condition here. The identical result on all ten seeds is the signature: with nothing to condition on, the model send out nearly the same small set of proposals every time. That is not a search finding a better optimum. It is a search that barely moves away from the reference region, and therefore one whose selection is not driven far by the validation score (on the reading of Section 4.3.6, is why its test error is low despite the worst validation error in the table).

4.3.3 Prompt and Model Capacity

Two explanations remain for the language model's showing: that the following prompt instructions, and does the stronger models help? Both can be tested directly, the first by rewriting the instruction to encourage exploration, the second by comparing across the size range available. Table 4.6 collects the outcome.

Table 4.6. Prompt variants: search behaviour. Distinct settings and grid coverage describe how widely a model searched.

Model	Prompt	Distinct	Coverage	Val. RMSE (m)	Test RMSE (m)
`llama3:8b`	default	9.1	0.41	0.1993 ± 0.0031	0.2453 ± 0.0028
	explore	19.9	0.68	0.1942 ± 0.0042	0.2422 ± 0.0057
`phi4:14b`	default	13.9	0.51	0.1994 ± 0.0049	0.2440 ± 0.0085
	explore	24.9	0.92	0.1929 ± 0.0029	0.2445 ± 0.0069
`nemotron:4b`	default	24.2	0.61	0.1945 ± 0.0033	0.2487 ± 0.0059
	explore	25.0	0.91	0.1894 ± 0.0045	0.2471 ± 0.0041
`gemma4-12B`	default	19.4	0.62	0.1935 ± 0.0063	0.2437 ± 0.0038
	explore	20.7	0.63	0.1930 ± 0.0042	0.2504 ± 0.0018
	OPRO	16.2	0.66	0.1896 ± 0.0041	0.2515 ± 0.0085
`qwen3-14B`	default	24.3	0.66	0.1996 ± 0.0061	0.2512 ± 0.0082
	explore	25.0	0.84	0.1972 ± 0.0057	0.2500 ± 0.0048
	OPRO	24.9	0.96	0.1943 ± 0.0055	0.2451 ± 0.0043
Random search	—	25.0	0.99	—	0.2413 ± 0.0099
Rule-based	—	25.0	0.61	0.1927 ± 0.0000	0.2371 ± 0.0000

The exploration prompt changes what the model does, and the size of the change

varies by model. It is largest where the default search was narrowest: `llama3:8b` more than doubled its distinct configurations, from 9.1 to 19.9, and raised its grid coverage from 0.41 to 0.68, while `phi4:14b` went from 13.9 to 24.9 distinct settings and from 0.51 to 0.92 coverage. Two models were already spending almost the whole budget on distinct settings and could barely move on that axis, `nemotron-3-nano:4b` from 24.2 to 25.0 and `qwen3-14B` from 24.3 to 25.0, both still widened the region those settings fall in, to 0.91 and 0.84 coverage. One model did not respond at all: `gemma4-12B` went from 19.4 to 20.7 distinct settings and from 0.62 to 0.63 coverage, so the instruction reached it without changing its search.

The accuracy was not affected. `phi4:14b` nearly doubles its distinct settings and coverage, and its test RMSE moves from 0.2440 to 0.2445 m. The three models that do improve, `llama3:8b`, `nemotron-3-nano:4b` and `qwen3-14B`, gain 0.0031, 0.0016 and 0.0012 m, every one of them smaller than that model's own standard deviation across searches. One model is made worse: `gemma4-12B` goes from 0.2437 to 0.2504 m, a loss of 0.0067 m, which is the largest effect the prompt has on any model in either direction. The last case is the informative one, because it's search the prompt did not widen at all. Its coverage moved by 0.01 while its error increased, which is the opposite of what was expected, and it was also the model with the best default result.

The OPRO-style prompt [17] presents the trial history as a sorted list of configurations and their numeric scores rather than as qualitative labels. It is the one variant that alters what the model is told rather than how it is instructed to behave, which is the distinction raised by research question 2 in Chapter 1. It was run on the two larger models, and it moves them in opposite directions: `qwen3-14B` improves from 0.2512 to 0.2451 m, while `gemma4-12B` degrades from 0.2437 to 0.2515 m. Whether showing the model raw numbers helps therefore depends on which model is reading them. It also produces the widest search any language model achieves here: `qwen3-14B` under OPRO covers 0.96 of the grid, above `phi4:14b`'s 0.92 under the exploration prompt and close to random search's 0.99, and its selection is still 0.0143 m worse than the fixed baseline.

Model scale explains no more than the prompt does. Ordered by size, the default results are 0.2487 m for the 4B model, 0.2453 for 8B, 0.2437 for 12B, 0.2440 and 0.2512 for the two 14B models. There is no monotone relation: the largest model tested is the worst of the five, and the spread across the whole range, 0.0075 m, is smaller than the gap between the best of them and the fixed baseline. Every configuration in Table 4.6 remains above the baseline's 0.2308 m and curve-aware rule-based 0.2264 m.

Taken together the two tests point the same way as the rest of this section. Where instructing the model to search more widely does change its search behavior, it changes its results negligibly, and where it changes nothing it still moves the result; using a larger model does not help either. What limits the LLM is not how much of the grid it visits or how large the model is, but the information on which it chooses among what it visits.

4.3.4 How the History Is Represented

The preceding experiments changed the model and the instruction. This one changes only how the same history is written down, which the protocol renders as qualitative labels and so strips of magnitude, spacing and the spread across trainings (Table C.1). Two

explanations compete. If that rendering is what limits the arm, restoring the numbers should improve its selections; if the limitation is what the model does with the history, a richer payload should change how it searches but not what it selects. They differ in a prediction: the first improves validation and test error together, the second stops at the validation objective. A third possibility concerns one block alone, that the best-so-far anchor invites imitation rather than inference, and predicts a wider search once it is removed.

Seven renderings were run, each altering one property with the grid, budget, seeds, scoring and validator frozen: labels, the control; numeric, measured RMSE, best epoch and gap in place of the labels; numeric_ci, those numbers plus the deviation over the three trainings and the three seed scores; ranks, ordering alone; no_anchor, the best-so-far block removed, which obliges a complete setting; no_conclusions, the mined patterns and derived labels removed; and window5, outcomes for the last five attempts only, with the already-tried list left complete so that repeats are not penalised.

Table 4.7. Each row changes one property of how the search history is written into the prompt. Δ is against that model's own labels control, negative better. Values/HP is the mean number of distinct values tried per hyperparameter, and Trained how many of the 25 attempts became evaluated settings. Rows marked † lost more than half their attempts to serving errors rather than to the model's own output.

Condition	Val. RMSE (m)	Test RMSE (m)	Δ test	Values/HP	Trained
`qwen3-14B`					
labels (control)	0.1996 ± 0.0061	0.2512 ± 0.0082	—	2.59	24.3
numeric	0.1930 ± 0.0038	0.2488 ± 0.0084	-0.0024	2.69	24.9
numeric_ci	0.1950 ± 0.0051	0.2514 ± 0.0075	$+0.0002$	2.59	24.7
no_anchor	0.1950 ± 0.0053	0.2456 ± 0.0038	-0.0056	4.01	25.0
no_conclusions	0.2002 ± 0.0022	0.2435 ± 0.0109	-0.0077	2.44	24.2
window5	0.1984 ± 0.0124	0.2494 ± 0.0088	-0.0018	2.49	24.6
ranks†	0.1970 ± 0.0069	0.2412 ± 0.0049	-0.0099	2.73	11.7
`gemma4-12B`					
labels (control)	0.1935 ± 0.0063	0.2437 ± 0.0038	—	2.53	19.4
numeric	0.1906 ± 0.0024	0.2514 ± 0.0082	$+0.0077$	2.69	22.7
numeric_ci	0.1904 ± 0.0018	0.2500 ± 0.0081	$+0.0063$	2.77	23.1
ranks	0.1901 ± 0.0012	0.2475 ± 0.0079	$+0.0038$	2.51	19.7
no_anchor	0.1942 ± 0.0007	0.2484 ± 0.0027	$+0.0047$	1.89	7.0
ranks†	0.1911 ± 0.0019	0.2542 ± 0.0104	$+0.0105$	2.33	8.4

The first explanation passes its first test and fails the second. On `qwen3-14B` every rendering that adds numbers lowers the validation error being minimised, by 0.0066 m for *numeric* and 0.0046 m for *numeric_ci* and *no_anchor* against the control's 0.1996 m, so the labels were indeed withholding something usable. The same renderings move test error by -0.0024, $+0.0002$ and -0.0056 m, each smaller than the control's own spread across searches (0.0082 m), and the validation ordering does not survive the change of split: *numeric* is the best validation cell and the third-best test cell. A richer history buys a better score on the split the arm may see, which Section 4.3.6 shows is not the same thing.

Making the payload poorer works better than enriching it. The best test cell is *no_conclusions* at 0.2435 m, then *no_anchor* at 0.2456 m, while numbers with their uncertainty give 0.2514 m, level with the control. The anchor case also confirms the third hypothesis: with no setting to take a delta against, the arm explored 4.01 distinct values per hyperparameter against the control's 2.59, the largest shift in search breadth in this chapter, on a full 25 trained attempts.

None of it transfers to the second model. On `gemma4-12B` all four usable variants are worse on test than its control, by 0.0038 to 0.0077 m, though three again improve validation, which extends to five renderings the model-dependence the OPRO row of Table 4.6 already showed. Two reservations attach to that block. Its control is itself thinned by the serving timeouts of Section 4.2.3, training 19.4 settings against the `qwen3-14B` control's 24.3, so each Δ there is measured against a weaker control and is if anything flattered. And `gemma4-12B` rejected 63 per cent of its own proposals when asked for a complete setting, training 7.0 of 25, so its *no_anchor* cell reports a budget rather than a representation.

One comparison escapes all of these, because it is made inside each run against the random and rule-based arms that ran beside the LLM and never saw a prompt: the model is worse in all thirteen cells, by 0.0008 to 0.0144 m against random search. Research question 2 is therefore answered in a qualified way. Performance does depend on the representation, measurably, but on the objective rather than the held-out error and with a model-dependent sign, and no rendering lifts the arm to the level of a method that reads no history at all. The label payload was consequently not what stood between this model and a good selection, since five other renderings of the same history, two of them carrying the raw numbers and their uncertainty, do not produce one either. The payload was varied one property at a time, so nothing here speaks to combinations of them.

4.3.5 Where the LLM Improves, Degrades and Stalls

The preceding experiments vary what the model is shown. This one holds the protocol fixed and investigates how the language model spends the budget it has: when it stops improving, and which of its proposals are worth making.

Table 4.8. How the language model's search behaves, per model. The last is the mean number of attempts for the final improvement. Early and late count improvements in attempts 1–8 and 9–25.

Model	Trained	Improvements	Last	Early	Late
`nemotron-3-nano:4b`	24.2	7.2	19.6	3.0	4.2
`gemma4-12B`	19.4	5.1	17.2	2.9	2.2
`phi4:14b`	13.9	4.5	14.5	2.4	2.1
`llama3:8b`	9.1	3.2	9.8	2.5	0.7

The models stall at very different points, and the ordering is the same as the ordering of their effective budgets. `nemotron-3-nano:4b`, which converts 24.2 of its 25 attempts

into trained configurations, is still improving at attempt 19.6 on average and makes more than half its improvements in the second half of the budget. `llama3:8b`, which converts only 9.1, has finished by attempt 9.8 and improves just 0.7 times after attempt 8. Every model makes a similar number of early improvements, between 2.4 and 3.0 , so what separates them is not how well they start but whether they have attempts left with which to continue. The stall is a consequence of the wasted budget documented in Section 4.2.3, not of an exhausted search space.

Across the four models, only 32.1% of the 608 trained attempts improved on the best score found so far; the remaining two-thirds cost three trainings each and changed nothing. Which change was proposed matters more than one might expect. Grouping every trained attempt by the hyperparameter it altered relative to the setting it was proposed against, and keeping the groups with at least ten instances, the ranges from 51.7% down to 10.7%: changing the hidden width improves the score half the time ($n = 60$), and changing patience half the time ($n = 24$), while changing the number of LSTM layers alone improves it in one attempt out of ten ($n = 28$). Weight decay (38.3%, $n = 149$) and dropout (24.3%, $n = 140$) are the two most frequently proposed changes and sit on opposite sides of the average.

The model's own reasoning is close to uninformative about this. Each proposal carries a stated diagnosis, and grouping attempts by that diagnosis gives improvement rates between 27.5% for plateau ($n = 102$) and 36.8% for possible underfitting ($n = 117$), against the overall rate of 32.1%. A nine-point spread around the base rate is a weak signal on which to act, and it is weaker than the spread across the hyperparameters themselves. The model is better at identifying which knob to turn than at explaining why, and it is the explanation, not the choice, that the qualitative payload is built to extract.

4.3.6 Selection Against a Noisy Validation Split

Every result reported so far is a test figure, but no method ever optimizes for it. The quantity each arm actually minimizes is the mean validation RMSE of Section 4.1.3, and the two are not in agreement.

Table 4.9. The objective the search minimizes against the metric it is judged by. Validation RMSE is the best score reached within the 25 attempts, averaged over the searches; test RMSE is the figure from Table 4.2. The rule-based controller is deterministic, hence the zero deviation.

Arm	Validation RMSE (m)	Test RMSE (m)
Fixed reference	≈ 0.294	**0.2308**
Rule-based	$\mathbf{0.1927 \pm 0.0000}$	0.2371
LLM (mean)	0.1968	0.2455

On the objective the search optimizes, the LLM and the rule-based controller are nearly tied (0.1968 against 0.1927 m) and both are far below the fixed reference's validation score of roughly 0.294 m. The LLM does find low-validation settings, roughly as well

as the controller does. Those same settings then land at 0.244–0.250 m on the held-out test block, worse than the reference's 0.2308 m. The proposals are therefore not poor on their own metric; what fails is the step from that metric to the one here reported. In this work, analysis is made by assumption and observation:

Observation: Selecting a configuration for a lower mean validation RMSE does not reliably produce a lower test RMSE. The ordering of the arms on validation is close to the reverse of their ordering on test, and the same dissociation appears again in the history experiment of Section 4.3.2, where every degradation of the payload raises the validation error and lowers the test error, and once more in the single-lever sweep of Section 5.2.5, where the validation score falls monotonically as the smoothness weight is raised while the test score does not follow. This is a measured property of the setup and it is what the comparison in Section 4.2 rests on.

Three mechanisms are consistent with that observation. The experiment cannot separate them, so none is adopted here.

The search may be fitting the noise of the validation block: 25 attempts scored on one 400-frame stretch, with the best of them kept. This is the mechanism Section 2.2.3 describes, and it is not specific to language models (Schneider et al. find that in roughly one HPO run in ten the selected configuration generalizes worse than the untuned default [11], which is the shape of the result in Section 4.2). It is also not specific to the arms that reason: Optuna, which carries no language model at all, shows the same divergence.

Validation and test sets are two contiguous stretches of one 2000-frame trajectory, so there is no reason to expect them to be exchangeable. If the person simply walks differently in one block than the other (more stop-and-go, a different mix of speeds) then a configuration better on validation can be worse on test, with no noise-fitting anywhere. On this reading the two scores measure different things rather than the same thing badly.

Each block is small, so each score carries its own substantial uncertainty, and part of the gap is measurement rather than any property of the configurations. Section 4.2.1 quantifies the corresponding spread on the test side: retraining one fixed configuration on the 30 evaluation seeds moves its RMSE by more than the differences between arms.

Distinguishing the three requires repeating the protocol against several validation splits, recordings, which Section 6.2 sets out and this study does not do.

The consequence for the thesis is that the comparison in Section 4.2 measures two things at once: how well a method optimizes the stated objective, and how well that objective predicts held-out error. On the first, the language model is competitive. On the second, no method is, and that is what the reported numbers reflect.

Chapter 5

Motion-Aware Optimization

Chapter 4 tested the language model on the nine hyperparameters, where whatever it knows about the world outside the training loop is of no use: nothing about a learning rate requires knowing how people move through a building. This chapter asks the question the thesis set out with. The architecture is frozen at the reference configuration and the model is asked to reshape the training objective instead, over a search space whose parameters describe motion (ceiling on plausible speed, a smoothness prior, and weights on slow, medium and fast movement) with a summary of how the tracked person actually moves supplied alongside the search history.

5.1 Motion-Aware Objective

The architecture, the data splits, the attempt budget, and the evaluation protocol are unchanged from Chapter 4. Only the object of the search changes: each method now controls six parameters of a composite loss. That loss has a base term.

In what follows $\hat{p}_t$ is the predicted position at time t and p_t the ground truth, both in the standardised coordinates the network trains in; σ is the per-coordinate standard deviation of the training targets, in metres, and f_s is the sampling rate. A standardised displacement becomes a displacement in metres on multiplication by σ, because the offset cancels in any difference, and it is this that makes Equation 5.1 dimensionally consistent.

$$\mathcal{L} = \underbrace{\frac{1}{B}\sum_{t} w_{r(t)}\,\|\hat{p}_t - p_t\|^2}_{\text{base (optionally regime-weighted)}} + \lambda_{\text{vel}}\,\frac{\overline{\text{relu}(\hat{v}_t - v_{\max})^2}}{v_{\text{ref}}^2} + \lambda_{\text{sm}}\,\frac{\overline{\|\hat{a}_t\|^2}}{a_{\text{ref}}^2}, \tag{5.1}$$

$$\hat{v}_t = \|(\hat{p}_t - p_{t-1}) \odot \sigma\| f_s, \qquad \hat{a}_t = (\hat{p}_t - 2p_{t-1} + p_{t-2}) \odot \sigma\, f_s^2. \tag{5.2}$$

Equation 5.2 gives the predicted speed and acceleration as one-step finite differences in SI units. The six searchable parameters are the max speed $v_{\max}$, the two penalty weights λ_{vel} and λ_{sm}, and the three regime weights w_{slow}, w_{med} and w_{fast}, applied to a sample according to the tercile $r(t)$ of its true speed. Setting $\lambda_{\text{vel}} = \lambda_{\text{sm}} = 0$ with uniform regime weights reduces Equation 5.1 exactly to plain mean-squared error.

Table 5.1 gives the discrete values each lever may take. The rule-based controller, random search, and the agent's validator all draw from these sets, so the methods differ only in how a vector is chosen. The neutral value of each lever is included, which is what makes plain mean-squared error a point inside the search space.

Table 5.1. The loss-shaping search space. The neutral (plain-MSE) vector is $v_{\max} = 2.0$, $\lambda_{\text{vel}} = \lambda_{\text{sm}} = 0$ and all three regime weights 1.0; it is the anchor every method starts from, so a proposal is a move away from no shaping. $v_{\max}$ has no effect while $\lambda_{\text{vel}} = 0$.

Lever	Allowed values	Units
$v_{\max}$	0.5, 0.75, 1.0, 1.5, 2.0	m/s
λ_{vel}	0, 0.05, 0.1, 0.2, 0.3	dimensionless
λ_{sm}	0, 0.05, 0.1, 0.2, 0.3	dimensionless
w_{slow}	0.5, 1.0, 1.5, 2.0, 3.0	dimensionless
w_{med}	0.5, 1.0, 1.5, 2.0, 3.0	dimensionless
w_{fast}	0.5, 1.0, 1.5, 2.0, 3.0	dimensionless

5.1.1 Units, Shapes and Normalisation

The conclusions of this chapter depend on the two penalties meaning the same thing on every recording.

All four position tensors in Equation 5.1 have shape $(B, 2)$ and hold standardised coordinates rather than metres, each stored as $(p - \mu)/\sigma$ with μ and σ taken from a standard scaler fitted on the training split alone. The σ of Equation 5.2 is that scaler's per-coordinate standard deviation: a fixed constant of the dataset, never learned. Table 5.2 gives its value on each recording, together with the sampling rate and the reference acceleration.

Table 5.2. Constants of each recording that enter Equation 5.1. σ is the per-coordinate standard deviation of the training targets, fitted on the training split alone, and converts a standardised displacement to metres. a_{ref} is the root-mean-square acceleration of the true training trajectory, computed at that recording's own sampling rate; the smoothness penalty is divided by its square.

Dataset	f_s (Hz)	σ (m)	a_{ref} (m/s^2)
Radar (`RadarEXP1`)	4	(0.806, 0.700)	1.412
Infrared (`IR-EXP2`)	5	(0.601, 0.585)	2.912

Figure 5.1 gives the per-sample step speed of both recordings. The two are not distinguished by their typical pace. Over the range holding almost every sample the infrared walker is the slower of the two, peaking lower and earlier, with a median step speed of 0.215 m/s against radar's 0.284 and a 95th percentile of 0.452 m/s against 0.600. Past the edge of that range the ordering reverses: its top speed is 5.56 m/s against radar's

3.34. That combination, a quieter body and a heavier tail, is what stands behind the factor of two in a_{ref}, and Section 5.2.4 reads the infrared results in its terms.

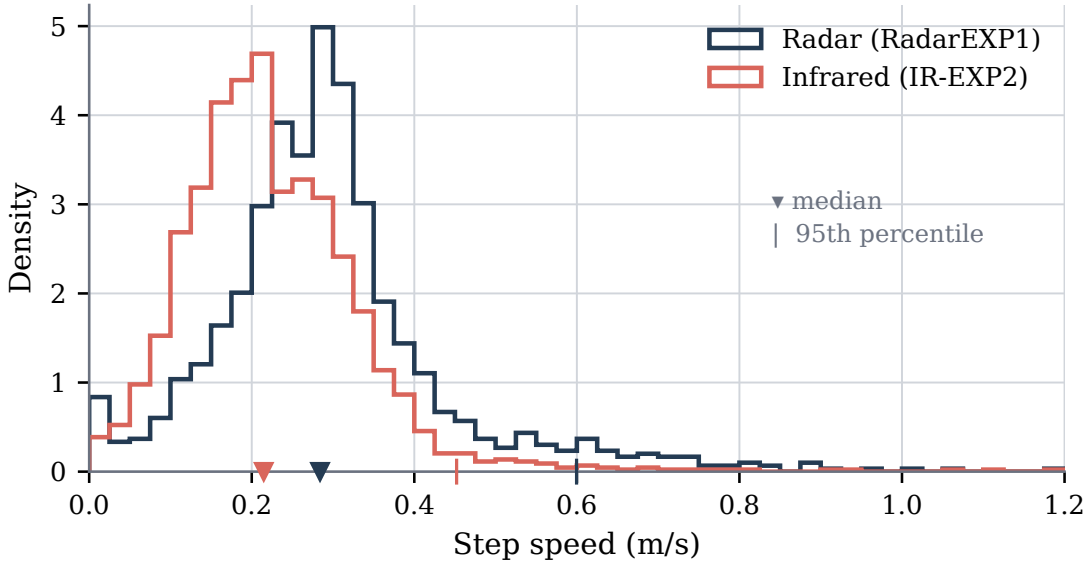


Figure 5.1. Step-speed distribution of each recording, on its training split. Speeds are per-sample step lengths converted to m/s at each recording's own sampling rate and measured on the training split alone, which is the trajectory, the rate and the split the motion summaries shown to the arms are computed from, so these are the quantities the arms were actually given.

Two properties follow: The first is comparability across recordings. Each penalty is divided by the corresponding reference kinematic, so that both weights are dimensionless and denote the same penalty strength relative to the walker's own motion at 4 and 5 Hz. The effect can be checked directly: with $\lambda_{\text{sm}} = 0.2$ the smoothness penalty evaluated at the natural-motion floor is 0.200 on all recordings, although a_{ref} spans a factor between them.

The second is temporal alignment. Each training sample carries its own p_{t-1} and p_{t-2} as fields alongside its input window, so alignment is a property of the sample rather than of its position in the batch, and shuffling the batches cannot break either kinematic term. The loss is computed per sample, not across a sequence.

A correction, and what it invalidated: An earlier implementation omitted the f_s^2 factor of Equation 5.2 and performed no cross-dataset normalisation. Because the term is squared, the smoothness penalty was wrong by a factor of f_s^4: 256 on radar and 625 on infrared. At the weights the grid offers, the prior was therefore doing almost nothing, and the same λ_{sm} meant different things on different sensors. The defect was not confined to a corner of the search space: between 83 and 99 per cent of the lever vectors those runs evaluated switched the smoothness term on (84% for qwen3-14B on radar, 94% for gemma4-12B on radar, 83% for qwen3-14B on infrared, and 99% for the no-profile control). Every result from those runs was therefore withdrawn, and the radar and

infrared experiments were repeated in full on the corrected loss, including the baseline and random-search methods.

5.1.2 Why this Smoothness Prior Is Misspecified

The basic idea of a smoothness prior is good. Our criticism here is not about that. People do move smoothly on short timescales: a walker's velocity changes over hundreds of milliseconds rather than between consecutive frames, so of two predicted trajectories that fit the sensor evidence equally well, the smoother one is the more plausible estimate of where the person actually was. Penalising implausible acceleration is a reasonable way to state that, it is the kind of physical regularity a learned model has no particular reason to discover on its own, and it is exactly the sort of domain knowledge Section 2.4 argues is worth supplying.

Both kinematic terms mix the current prediction with previous ground-truth positions, p_{t-1} and p_{t-2}. For the velocity term this is harmless: $\mathrm{relu}(\hat{v}_t - v_{\max})^2$ is a dependent, zero for any predicted step slower than the maximum, so it forbids a region rather than pulling toward a target and contributes no gradient at a physically plausible prediction.

The smoothness term does not have that property. The second difference measures how sharply the path bends at time t, but p_{t-1} and p_{t-2} are fixed ground truth, so the only free quantity is the prediction, and the term takes to a question: does the prediction continue the previous two points in a straight line? It is zero at the constant-velocity extrapolation $\hat{p}_t = 2p_{t-1} - p_{t-2}$.

That point is not the truth. Whenever the person turns or changes pace, the true position departs from the straight-line continuation, and the penalty is then positive at the correct answer with a gradient pointing away from it. The term fines the model for the turns the person actually made. Table 5.3 quantifies the displacement. Sweeping the prediction from the true position toward the extrapolation and locating the minimum of the total loss places that minimum between 88 and 98 per cent of the way to the extrapolation on radar, and between 73 and 94 per cent on infrared, across the non-zero weights the grid offers.

The absolute displacement is modest, which is why training does not collapse: truth and extrapolation lie 0.055 m apart on radar and 0.040 m on infrared, against model RMSEs of roughly 0.23 and 0.21 m. But the optimum of the shaped objective is not the truth, and that is not a property a prior should have.

Table 5.3. Where the total loss is minimised, expressed as a fraction of the distance from the true position to the constant-velocity extrapolation $2p_{t-1} - p_{t-2}$, for the non-zero smoothness weights in the grid. Computed on the training split of each recording.

λ_{sm}	Radar	Infrared
0.05 (lightest non-zero)	88%	73%
0.1	94%	84%
0.3 (grid maximum)	98%	94%

The natural repair is to make the acceleration term dependent as well, penalising only acceleration above a physical ceiling. Real turns would then cost nothing, the term would have a zero floor, and no bias toward straight lines would remain. It was not implemented here: folding a redesign of the term into a correction of its units would make the two inseparable in the results. Section 6.2 takes it up.

What the term does in its present form is better described as shrinkage (a pull toward smooth motion that trades bias for variance and damps sensor noise). Offered as an interpretation, this matters because shrinkage is a generic smoother and not a statement about human movement. Its value should then fall as the motion content of a recording rises, and the harm should concentrate where the path turns. Section 5.2.5 tests both predictions on the lever alone, with everything else held fixed, and finds them supported: sweeping the weight lowers test error slightly on radar and raises it on infrared, the recording with the sharper turns and twice the reference acceleration, and on infrared the cost falls specifically in the sharp-turn bin. Section 5.2.4 shows the same contrast at the level of whole methods.

No test leakage: Every quantity the loss depends on is fitted on training data and then frozen: the scaler σ, the tercile edges that define the speed regimes, and the reference kinematics v_{ref} and a_{ref}. The p_{t-1} and p_{t-2} a term uses are the ground-truth positions of the same split as the sample, so a training batch draws only on training positions and a validation batch only on validation positions. The motion profile shown to the language model (the speed, turn and dwell summaries) is computed on the training split. Configurations are selected on validation RMSE, and the test set is read once, after selection, exactly as in Chapter 4.

5.2 Results

5.2.1 Arms and Controls

The comparison is built so that the model's knowledge of motion can be separated from the extra flexibility a six-parameter objective provides. An improvement over plain mean-squared error is not by itself evidence of anything about movement, since the richer objective has more ways to fit the data however its levers are set.

- **Plain MSE**: all penalty weights zero and uniform regime weights. Deterministic, and the reference that the other methods are measured against.
- **Motion rule-based**: a deterministic controller that reads the motion profile once and proposes a single lever vector. It performs no search at all.
- **Random over levers**: random search of the same six levers, which isolates the flexibility of the objective from any reasoning about it.
- **LLM, levers only**: the LLM with the per-regime error breakdown, and the motion summary withheld.
- **LLM + motion profile**: the LLM with all profile.

5.2.2 Radar

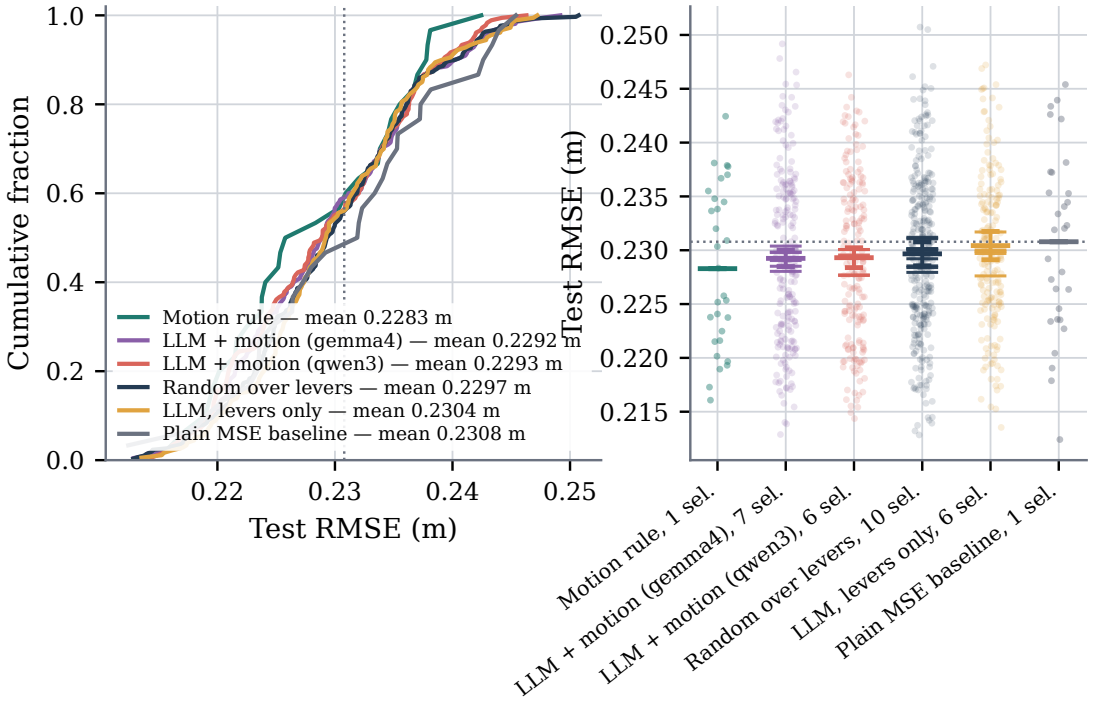


Figure 5.2. Every motion arm on the radar recording. Left: cumulative distributions of test RMSE, with repeated selections counted once. Right: the same evaluations as points, with the mean and standard deviation taken across searches; the long ticks mark the per-selection means. The label k sel. gives the number of distinct lever vectors an methods searches selected, so $k = 1$ marks a deterministic arm and a larger k produces a correspondingly denser cloud. The dotted line marks the plain-MSE baseline.

Table 5.4. Motion-aware loss shaping on the radar recording. Test RMSE is the mean over 30 evaluation seeds; ± is the standard deviation across searches, and is zero by construction for the two deterministic arms. Lower is better.

Arm	Test RMSE (m)
Motion rule	0.2283
LLM + motion profile (`gemma4-12B`)	0.2292 ± 0.0008
LLM + motion profile (`qwen3-14B`)	0.2293 ± 0.0009
Random over the same levers	0.2297 ± 0.0011
LLM, levers only (motion summary withheld)	0.2304 ± 0.0013
Plain MSE (no shaping)	0.2308

Loss shaping helps on radar. Every method that tunes or fixes the six levers lands below the plain-MSE baseline of 0.2308 m. The ordering, however, is unfavorable to the

hypothesis this chapter tests. The best is the deterministic motion rule-based at 0.2283 m, which reads the profile once and proposes a single lever vector without searching at all. The two language-model follow at 0.2292 and 0.2293 m, and random search over the same six levers reaches 0.2297 m.

The differences are of the same order as the seed-to-seed spread, so the means alone cannot settle them. Each selection is therefore paired seed by seed against its comparator, over the search seeds the two methods share; Table 5.5 reports the result.

Table 5.5. Paired differences between arms on the radar recording, computed over the search seeds the two arms share. Negative means the first arm is better; * marks a 95% confidence interval excluding zero, and first better counts the shared seeds on which the first arm's selection wins. The motion rule and baseline are both deterministic, so their difference has no across-seed variation and no interval is reported for it.

Comparison	Difference (m)	95% CI	t	First better
Motion rule − baseline MSE	-0.0025^{*}	(deterministic)	—	30/30
LLM (qwen3) − baseline MSE	-0.0016^{*}	$[-0.0023, -0.0008]$	-5.02	30/30
LLM (gemma4) − baseline MSE	-0.0015^{*}	$[-0.0022, -0.0009]$	-5.77	30/30
Random levers − baseline MSE	-0.0009^{*}	$[-0.0017, -0.0002]$	-2.84	21/30
LLM, levers only − baseline MSE	-0.0003	$[-0.0014, +0.0008]$	-0.64	16/30
LLM (qwen3) − motion rule	$+0.0009^{*}$	$[+0.0002, +0.0017]$	$+2.99$	6/30
LLM (gemma4) − motion rule	$+0.0010^{*}$	$[+0.0003, +0.0016]$	$+3.56$	4/30
LLM (qwen3) − random levers	-0.0004	$[-0.0015, +0.0008]$	-0.69	21/30
LLM, levers only − LLM (qwen3)	$+0.0011^{*}$	$[+0.0001, +0.0022]$	$+2.45$	12/30

Three findings follow, and they point in different directions.

The motion summary contributes: Withholding the motion summary, while leaving the per-regime error breakdown in place, costs 0.0011 m ($t = +2.45$, interval excluding zero), and the arm deprived of it becomes statistically indistinguishable from baseline mean-squared error: -0.0003 m, $t = -0.64$, better on 16 seeds. The language model's entire advantage over baseline MSE on radar therefore depends on being shown how the person moves. This is the one place in the study where the model's motion input demonstrably does work, and it supports the hypothesis set out with.

But a motion rule-based reading the same profile beats it: Both language-model arms have slightly lower value than the motion rule, by $+0.0009$ and $+0.0010$ m, better on 6 and 4 seeds. The motion rule-based model performs no search whatever: profile in, one lever vector out. Whatever the model adds by iterating over attempts does not recover the gap.

It is indistinguishable from random search: Random sampling of the same six levers differs from the language model by -0.0004 m, with an interval spanning zero ($t = -0.69$). The extra degrees of freedom, rather than any reasoning over them, therefore account for what the model achieves. Motion loss-shaping helps on radar.

5.2.3 Where the Gain Comes From

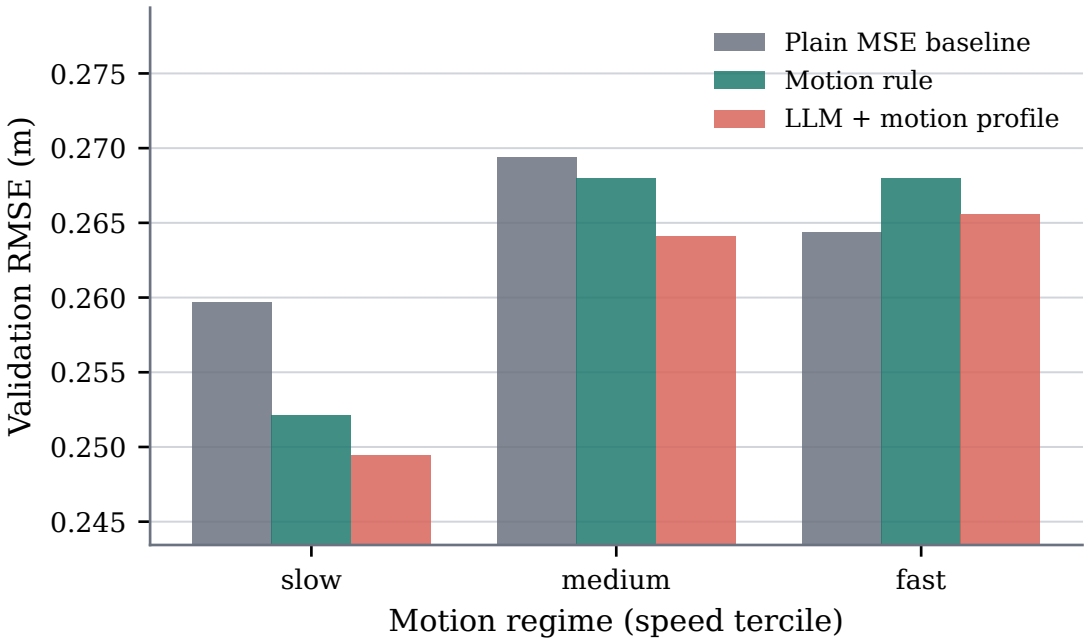


Figure 5.3. Validation RMSE by speed regime on the radar recording, averaged over searches. The bins are computed from the true speed of each sample, on the training split. The gain from loss shaping falls in the slow and fast regime, which the objective's bin weights exist to address, is where the baseline is beaten.

The name of the objective suggests where its gain ought to fall. Figure 5.3 tests that directly, breaking validation RMSE down by the speed-regime of the sample. The baseline is worst in the medium regime, at 0.2694 m, rather than in the fast one at 0.2644 m: error does not rise monotonically with speed. The language model improves the slow regime by 0.0103 m and the medium regime by 0.0053 m, but leaves the fast regime 0.0012 m worse than the baseline, and the motion rule-based shows the same pattern better. Section 5.1.2 supplies the reading: a penalty that pulls predictions toward constant-velocity motion helps where the true motion is nearly constant, and stops helping, where it is not.

The lever values the selected methods in the same way, from independent evidence. Of the 223 vectors `qwen3-14B` had accepted, 113 set the smoothness weight to 0.05, the lightest non-zero value in the grid, and one set it to the grid maximum of 0.3; `gemma4-12B` put 88 of its 159 at 0.1. That a light penalty is preferred is what Table 5.3 predicts. The contrast with the control is the striking part: the method from which the motion summary was withheld put 160 of its 224 vectors at 0.2 or 0.3, and it is the worst-performing shaping. Shown how the person moves, the model turns the misspecified prior down; Without that information, it turns it up.

5.2.4 A Second Sensor: IR

The radar result rests on a single recording, which cannot establish that loss shaping is a technique a practitioner could adopt. The whole experiment was therefore repeated on an infrared recording of a different movement character. That recording is 8817 frames at 5 Hz of a sixteen-channel infrared array, with two target columns, the planar position of the tracked person in meters. The window is 1 second, that is 5 frames. Figure 5.4 shows the ground-truth path in each partition, on the same conventions as Figure 4.2 for radar. The structure of the three blocks is worth noting: the training block is a dense raster of horizontal sweeps, while the validation block is a perimeter walk with two diagonals, so the two are not samples from a single walking behavior. That is a property of the recording rather than a choice made here, since the split is fixed by the protocol, but it is worth keeping in view alongside the disagreement between validation and test error that Section 4.3.6 documents on radar.

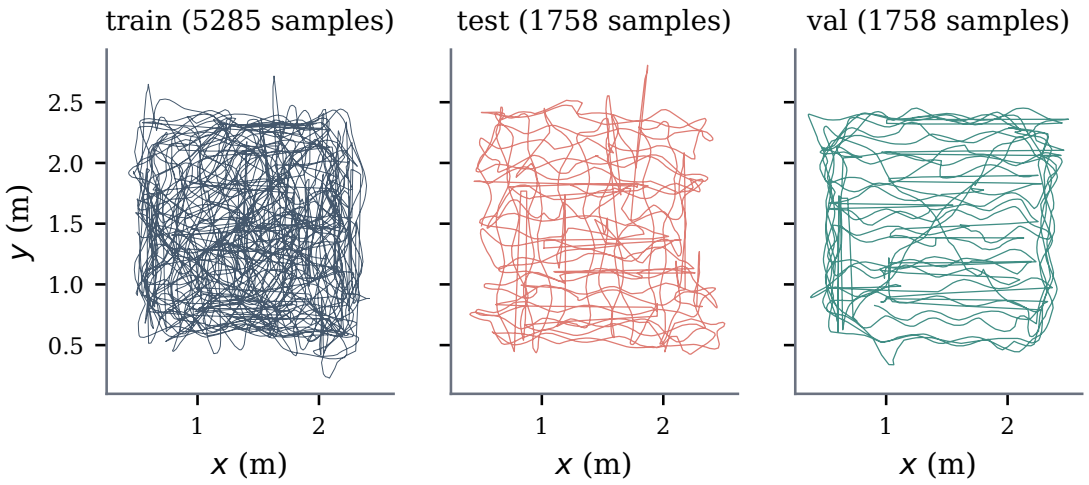


Figure 5.4. Ground-truth trajectory in each partition of the infrared recording, on a common pair of axes.

The contrast that matters is not the typical speed (the infrared walker is slower most of the time, with a 95th-percentile speed of 0.45 m/s against radar's 0.60) but the shape of the distribution. Its top speed is 5.56 m/s against radar's 3.34, so it has rarer but much faster excursions, and its reference acceleration is more than twice as large, at 2.912 against 1.412 m/s^2. Both facts point the same way: the infrared recording carries more genuine motion content for a smoothness before fight, and a heavier tail for a speed ceiling to clip.

The two methods that read the motion profile change sign. On radar, the motion rule-based was the best one, at −0.0025 m; on infrared, it is the worst, at +0.0025 m and beaten by baseline MSE. The `qwen3-14B` model moves from −0.0016 to +0.0011 m, and `gemma4-12B` lands on the line at +0.0003 m, with an interval spanning zero.

The arm that does not read the profile does not change sign. Random search over

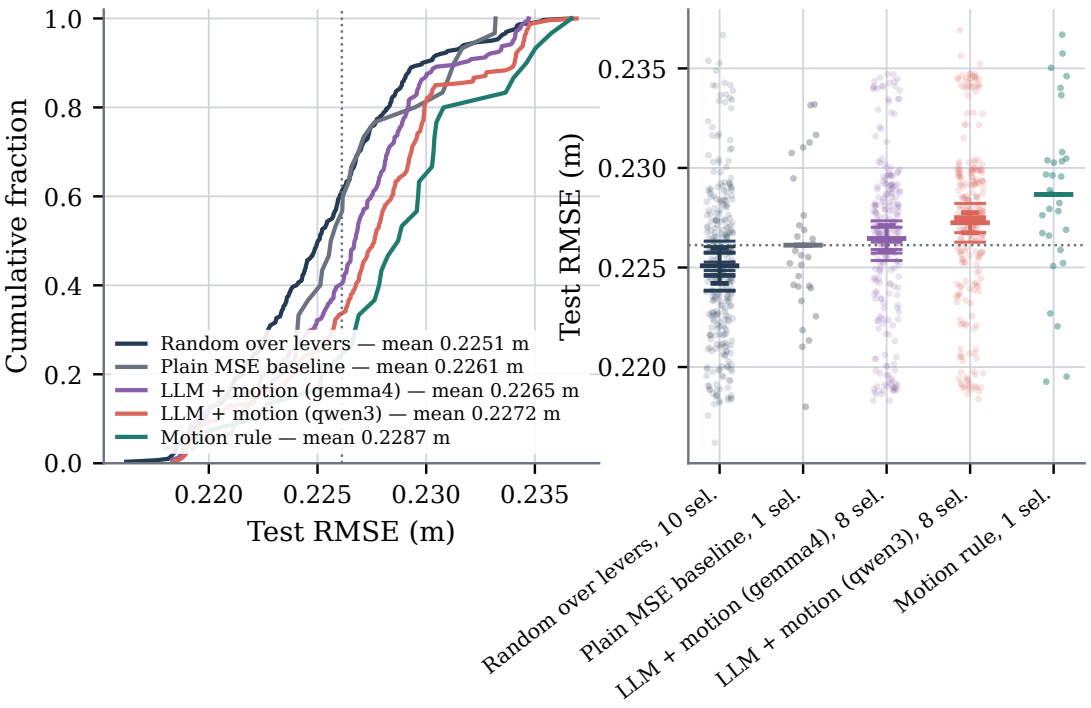


Figure 5.5. The same comparison, and the same conventions, as Figure 5.2, on the infrared recording. The arms are ordered by mean in both figures, so the change in ordering is the visual form of this subsection's point: the plain-MSE dotted line now sits second from the left rather than last, and only one arm lies below it.

Table 5.6. Motion-aware loss shaping on the infrared recording, each arm paired seed by seed against baseline MSE. Negative means the arm beats plain MSE; * marks a 95% confidence interval excluding zero, and better on counts the search seeds on which the arm's selection wins.

Arm	Test RMSE (m)	vs. plain MSE	Better on
Random over levers	**0.2251** ± 0.0009	−0.0010*	27/30
Baseline MSE	0.2261	—	—
LLM + motion profile (`gemma4-12B`)	0.2265 ± 0.0007	+0.0003	9/30
LLM + motion profile (`qwen3-14B`)	0.2272 ± 0.0005	+0.0011*	0/30
Motion rule-based	0.2287	+0.0025*	0/30

the same six levers helps on both recordings (−0.0009 m on radar and −0.0010 m on infrared, ($t = -3.71$)) and it is the best arm on infrared.

The ordering within the dataset also flips, which has to be said because it contradicts radar. On infrared, both language-model beat the motion rule-based (by −0.0014 m for `qwen3-14B` ($t = -8.90$) and −0.0022 m for `gemma4-12B` ($t = -10.71$)). Neither ordering is stable across sensors. What is stable is that the language model never beats random search over the same levers: on radar it ties it, and on infrared it loses to it by +0.0022 m.

The selected lever vectors say why, and they are evidence independent of the scores. On infrared, every arm shown the motion profile set the speed ceiling to 0.5 m/s in all of its searches, while random search spread its choices between 0.5 and 1.5. The mean selected smoothness weight tells a similar story: 0.220 for `qwen3-14B` and 0.230 for `gemma4-12B`. Random search selected 0.125, and switched the velocity penalty off entirely.

The infrared data the arms were shown is scattered: mean speed 0.29 m/s, 95th percentile 0.45 m/s, maximum 5.56 m/s. Read as a description of the operating range, the percentile says that this person moves at about half a metre per second, and a ceiling at 0.5 m/s. Every arm that saw the profile drew exactly that conclusion. The ceiling then penalises the deviations to 5.56 m/s, which are the samples a localizer is most likely to get wrong, and the one arm that escaped is the one that ignored the profile altogether.

On infrared, the motion summary was used consistently (Section 5.2.7 measures the coupling between stated reason and action at 96 to 98 per cent) using it consistently is what produced the worse setting. Information that is used is not thereby useful. This evidence comes from the settings a joint six-lever search happened to select rather than from a controlled sweep, so it identifies a mechanism without establishing one; Section 6.2 sets out the experiment that would.

Two objections have been raised against this reversal. The first is that it might be an artefact of the unit defect of Section 5.1.1, since infrared is the recording that defect distorted most, by a factor of $f_s^4 = 625$ against radar's 256. The experiment was repeated in full on the corrected loss. The second is that those repeated runs still carried radar's window length and sampling rate on the infrared data, because the dataset specification was not re-applied when the recording was changed. The experiment was repeated, at the correct window of five samples at 5 Hz. Table 5.6 and Figure 5.5 are that second re-run, and the earlier infrared numbers are withdrawn along with the pre-correction ones. The reversal survived, but it narrowed in the way reported above: random search over the levers now helps rather than hurts.

Section 5.1.2 supplies a mechanism for the smoothness half of the effect, while the lever evidence above adds the velocity half, which that argument did not anticipate. The radar regime breakdown of Section 5.2.3 is the same phenomenon within a single dataset. The two are independent and they agree.

5.2.5 The Smoothness Prior in Isolation

Everything reported so far is an argmin: the single best-of-25 lever vector each search selected. That says what the methods chose, but nothing about the shape of the response to any one lever, because the other five move at the same time. Since Section 5.1.2 argues that one lever in particular, the smoothness weight, is misspecified, and since that argument predicts something about which values should be preferred, both sides are measuring directly. This subsection does so in two steps: what the methods selected, and what a controlled sweep of the lever alone produces.

Which values were selected: The grid offers five values, $\lambda_{\text{sm}} \in \{0, 0.05, 0.1, 0.2, 0.3\}$ (Table 5.1), of which 0 switches the term off. Table 5.7 gives what each arm chose, on both recordings.

Table 5.7. The smoothness weight selected by each arm, over its complete searches. Weak counts the searches selecting 0.05 or 0.1, strong those selecting 0.2 or 0.3, and off those selecting 0.

Recording	Arm	Mean λ_{sm}	Off	Weak	Strong
Radar	Motion rule-based	0.050	0	9	0
	LLM + profile (`qwen3-14B`)	0.075	0	10	0
	LLM + profile (`gemma4-12B`)	0.125	0	6	2
	Random over levers	0.160	2	3	5
	LLM, levers only (no profile)	0.255	0	2	8
Infrared	Motion rule-based	0.050	0	10	0
	Random over levers	0.125	0	7	3
	LLM + profile (`qwen3-14B`)	0.220	0	1	9
	LLM + profile (`gemma4-12B`)	0.230	0	0	10

Three things follow. Weak values are systematically preferred on radar by every arm that reads the motion profile, and on that recording the ordering by mean selected weight tracks the ordering by test accuracy in Table 5.4, from the best arm at 0.050 to the worst shaping arm at 0.255; the only inversion is between the two language models, which differ by 0.0001 m. The preferred strength changes between recordings, and it changes only for the arms that read the profile: they move from 0.075 and 0.125 on radar to 0.220 and 0.230 on infrared, between doubling and tripling it, while the deterministic rule stays at 0.05 by construction and random search moves the other way, from 0.160 to 0.125. No method shown the motion profile ever selected 0 on either recording; the only searches that switched the prior off were two random ones on radar.

What the lever does on its own: Selections are confounded, so the sweep of Table 5.8 removes the confound: the nine hyperparameters stay frozen at the reference configuration, the other five levers stay neutral, and only λ_{sm} moves across its full grid. The $\lambda_{\text{sm}} = 0$ cell is then plain mean-squared error, so the baseline sits inside the sweep instead of being imported, and every cell is trained on the same evaluation seeds, so the comparison against that cell is paired.

Table 5.8. Sweeping λ_{sm} with the nine hyperparameters frozen at the reference configuration and the other five levers neutral. Validation RMSE is the search-side score over the three scoring seeds. Δ is that paired difference, negative meaning the prior helped.

	Radar			Infrared		
λ_{sm}	Val. RMSE	Test RMSE	Δ (t)	Val. RMSE	Test RMSE	Δ (t)
0 (plain MSE)	0.2251	0.2359	—	0.2094	0.2252	—
0.05	0.2176	0.2354	−0.0005 (−0.17)	0.2094	0.2262	+0.0009 (+0.63)
0.1	0.2148	0.2347	−0.0012 (−0.59)	0.2087	0.2269	+0.0017 (+1.18)
0.2	0.2122	0.2349	−0.0009 (−0.43)	0.2087	0.2267	+0.0015 (+1.11)
0.3	0.2126	0.2326	−0.0033 (−1.42)	0.2086	0.2267	+0.0015 (+1.02)

Read alone, the lever is weak on radar and slightly harmful on infrared. Every non-zero value lowers radar test error and every non-zero value raises infrared test error, so the honest summary is that in isolation the prior is close to inert on radar and slightly costly on infrared. The response is also flat rather than peaked: on radar there is no interior optimum, and on infrared the penalty is paid almost in full at the lightest weight and then stops growing.

Two further readings matter. The first is that the validation score falls monotonically with λ_{sm} on both recordings while the test score does not follow, which is the split of Section 4.3.6 reproduced in a setting with a single moving parameter and no search at all. Whatever produces that gap, it is not specific to the arms or their proposals. The second is a caution on absolute levels. A 0.004 m run-to-run difference at fixed seed is larger than most of the effects this chapter reports, which is why the Δ column, paired inside one run, is the only column that should be compared with the tables above.

Where the prior helps and where it hurts: The bias argument of Section 5.1.2 makes a sharper prediction than an overall mean can test: a term minimised at the constant-velocity extrapolation should help where the path is straight and hurt where it turns. Table 5.9 breaks the sweep's test error down by the heading change of the true trajectory.

Table 5.9. Test error of the sweep by the heading change of the true trajectory, as the paired difference from the $\lambda_{\mathrm{sm}} = 0$ cell; negative means the prior helped that bin. Errors here are mean 2D Euclidean distances.

	Radar		Infrared	
λ_{sm}	Straight	Sharp turn	Straight	Sharp turn
0.05	−0.0081	−0.0024	−0.0047	+0.0016
0.1	−0.0102	−0.0026	−0.0042	+0.0030
0.2	−0.0107	−0.0046	−0.0033	+0.0020
0.3	−0.0110	−0.0069	−0.0033	+0.0020

The predicted signature appears on infrared and not on radar. On infrared the prior helps the straight bin at every weight and hurts the sharp-turn bin at every weight, which is what a term that fines the model for real turns should do. On radar it helps both bins, straights roughly twice as much as turns, so there the term behaves as a mild generic smoother and the bias is not large enough to show. That the two recordings differ this way is consistent with their motion content: the infrared walker turns less often but far more sharply when it does (6.4% of samples against radar's 20.1%, but the two recordings reach 95th-percentile heading changes of 56 and 114 degrees on very different sampling intervals), and its reference acceleration is more than twice as large.

What this settles, and what it does not: Smoothness is a physically reasonable prior for human motion, and nothing here argues otherwise. What the sweep supports is narrower and concerns this implementation: mixing the current prediction with two fixed ground-truth positions produces a term whose optimum is the constant-velocity

continuation rather than the truth, and on the recording with the sharper turns that shows up exactly where the argument says it should. Whether a correctly specified smoothness prior would help is not answered by these runs and is not implied by them. What can be said is that the form used here is not a good test of the idea, and that on the recording where its defect bites the arms which read the motion profile made the wrong call about it: the sweep puts the best infrared cell at $\lambda_{\text{sm}} = 0$ and the cost roughly flat above it, and those arms selected 0.2 or 0.3 in nineteen of their twenty searches, while random search, which never saw the profile, stayed at 0.05 or 0.1 in seven of ten.

5.2.6 Run Integrity

A few facts about how the searches actually ran should be recorded, so that no number above is read as a cleaner result than it is. On radar, `qwen3-14B` had 223 of its 250 attempts accepted (177 clean and 46 after a single retry) with 27 rejected, of which 21 were settings already tried. The no-summary control accepted 224 of 250, and no payload in it contained a motion summary. `gemma4-12B` accepted 159 of 200; of its 41 rejections, 39 were lost to the 300-second per-call timeout, which makes its cell the least well searched.

The infrared runs behind Section 5.2.4 are the better-behaved half of the experiment, that is where the negative result falls: `qwen3-14B` accepted 240 of 250 (204 clean and 36 after a retry) and `gemma4-12B` 225 of 250 (187 clean and 38 after a retry). No arm lost a seed, and the timeout problem that thinned `gemma4-12B`'s radar cell was halved rather than removed: 23 of that arm's 25 infrared rejections were still per-call timeouts, 9 per cent of its calls, compared with 19 per cent on radar.

5.2.7 Does the LLM Reason, or Label?

A clean, in-grid proposal shows that the model follows the protocol, it does not show that the model reasons about the problem. Two measurements were done in this research:

In the generic search, the diagnosis is a formatting token. Every proposal records the diagnosis the model assigned before proposing a change, so the two can be compared. The test can be stated in advance. If the diagnosis were a genuine reading of the training state it should vary as that state varies, and it should determine the direction of the change, if it were a formatting token it would sit at a near-constant value and bear little relation to the action that follows.

Table 5.10 shows that two of the four models collapse onto a single label: `llama3:8b` calls the state possible underfitting on 91 per cent of the replies it diagnoses at all, and `phi4:14b` calls it healthy on 86 per cent. The other two vary more (`nemotron-3-nano:4b` says possible overfitting on 71 per cent, and `gemma4-12B` spreads across five labels with none above 42 per cent) but the extremes are the informative cases, since a diagnosis that almost never changes cannot be read as a state that does.

The direction of the change sharpens the point. Having diagnosed underfitting, `llama3:8b` raises weight decay on 221 of the 325 such proposals that touch it, lowering it on 102 and leaving it unchanged on 2: more regularisation, which is the standard

response to overfitting. Diagnosis and action point in opposite directions. The incoherence is not universal, `nemotron-3-nano:4b`, on the same diagnosis, lowers weight decay on eight of the ten proposals that name it, which is the direction the diagnosis implies, though on too few proposals to carry much weight. `phi4:14b` shows the complementary inconsistency: having called the state healthy, which implies that no change is needed, it proposes a change on every one of those 347 replies, and `gemma4-12B` does the same on all 40 of its own. The label reads as a fixed preamble proposed to satisfy the response format, not as an assessment that drives the decision.

Table 5.10. The diagnosis each model assigned before proposing a change, over the generic hyperparameter search. Replies counts every reply the model returned, retry replies included; with diagnosis counts how many of those carried a parsable `diagnosis` line, and the share is of that second column rather than the first.

Model	Replies	With diagnosis	Most frequent diagnosis	Share
`llama3:8b`	434	434	possible underfitting	91%
`phi4:14b`	405	405	healthy	86%
`nemotron-3-nano:4b`	319	314	possible overfitting	71%
`gemma4-12B`	335	181	plateau	42%

In the motion search, the stated reason tracks a concrete quantity. Here the prompt supplies a specific number (the error in each speed regime). Across the accepted proposals of the `qwen3-14B` arm, 96 per cent name a specific motion regime, and every one of those up-weights a regime it named. Reacting to a quantity, the model is consistent; asked to name a state, it is not.

The point is what makes the observation defensible. The no-summary control also receives the per-regime error breakdown and shows the same coupling (98 per cent name a regime, and all of those up-weight it) while performing no better than baseline MSE. Consistency between stated reason and action is therefore a property of having been given a concrete number to react to, and not evidence that the resulting decisions are good. It is a statement about the input, not about the reasoning.

5.3 What the Motion Experiment Establishes

The architecture, the data splits, the attempt budget and the evaluation protocol were held at the values of Chapter 4, and only the object of the search changed: from nine conventional hyperparameters to six parameters of a composite loss. With every lever at its neutral value that loss reduces to baseline MSE, so it is the zero point of the same search, and every comparison in this chapter is paired within one search space.

On radar, loss shaping helps. Every method that tunes or fixes the six levers lands below the MSE baseline of 0.2308 m, and the motion summary is part of why: withholding it, while leaving the per-regime error breakdown in place, costs 0.0011 m ($t = +2.45$), and the arm deprived of it becomes indistinguishable from baseline ($t = -0.64$). This is

the place in the study where the model's motion input demonstrably does work, and it supports the hypothesis the thesis set out with.

Nothing about it, however, requires a language model. A deterministic controller that reads the same profile once and emits a single lever vector, searching not at all, beats both language-model arms on radar by +0.0009 and +0.0010 m. Random sampling of the same six levers is statistically indistinguishable from the model (−0.0004 m, $t = -0.69$). The extra degrees of freedom, rather than any reasoning over them, account for most of what the model's arm achieves which is what the control arms were introduced to expose.

Nor does it transfer. On infrared the two arms that read the motion profile change sign: the rule goes from the best arm to the worst (+0.0025 m, beaten on ten seeds out of ten) and `qwen3-14B` from −0.0016 to +0.0011 m, also ten of ten. The arm that ignores the profile does not change sign, helps on both recordings, and is the best arm on infrared. Section 5.2.4 locates the mechanism: every motion-informed arm read the infrared profile's 95th-percentile speed of 0.45 m/s and set the ceiling at 0.5 m/s in ten searches of ten, against genuine excursions to 5.56 m/s, while the one arm that escaped switched the velocity penalty off entirely. The motion summary was used consistently on infrared (Section 5.2.7 measures the coupling at 96 to 98 per cent) and using it consistently is what produced the worse setting. Information that is used is not thereby useful.

The answer to the question this chapter asked is therefore mixed, and worth stating precisely. The six-parameter objective contains configurations that can improve on the plain MSE baseline on both recordings, but the benefit of supplying motion information to the language model does not transfer to the second recording. The statement the evidence supports is therefore neither that the model exploits human-motion knowledge nor that such knowledge has no effect, but the one in between: motion information influences the model's choices and can improve them, but it is not established that the model uses that information better than random search over the same levers or than a simple deterministic heuristic, and the improvement does not carry over to the second recording.

Three boundaries qualify even that. The study covers two recordings and one architecture, and its seeds vary training randomness rather than subjects, rooms or sensors (Section 4.1.5). The smoothness prior is misspecified, as Section 5.1.2 argues and Section 5.2.5 measures, so what was tested is one flawed instantiation of motion-aware training rather than the idea in general, and the hinge repair is deferred to Section 6.2. And the per-regime and lever-choice evidence comes from a joint six-lever search, so it identifies mechanisms rather than proving them; the single-lever sweep of Section 5.2.5 closes that gap for one lever only. None of these touches the negative result about the language model, which does not depend on the prior being well specified: every arm searched the same flawed objective, and the motion rule-based and random search matched the language model.

Chapter 6

Conclusion

This thesis asked whether a large language model can serve as the search strategy for hyperparameter optimization of a compact LSTM indoor localizer, and whether reasoning over a history of trials offers any advantage over methods that do not reason at all. To answer it, a single-agent LLM optimizer was built and placed under a protocol that removes the usual escape routes: an identical discrete grid, an identical budget of 25 attempts, identical scoring by mean validation RMSE over three fixed seeds, strict validation that rejects any proposal the other methods could not also have made, and a final evaluation on seeds disjoint from everything used during search.

Under that protocol the answer is negative on both counts. Across all searches that span four open models, LLM selected configurations averaging 0.2455 m test RMSE, worse than random search (0.2413 m), the rule-based controller (0.2371 m), and fixed reference (0.2308 m). It beat only Optuna. On this payload no search method achieved a lower mean test RMSE than the fixed baseline in the original comparison of the nine hyperparameters. When the payload is enriched with a per-epoch training-curve summary, the deterministic rule-based controller reaches 0.2264 m (Section 4.3.1).

Three findings give that headline more shape. First, the LLM's weakness is not that it explores too little. It does explore less, evaluating 16.6 distinct configurations against 25 for random search and losing up to two-thirds of its budget to rejected repeat proposals, but the model that explored most finished last and the model that explored least finished mid-field. The limitation lies at least as much in which configurations are chosen as in how many. Second, the LLM is the most consistent of the stochastic methods, with a standard deviation of 0.0058 m against random search's 0.0099 m, which means it converges reliably to a mediocre region instead of occasionally reaching a good one. For a search method, consistency without quality is of little use. Third, sample efficiency and final accuracy proved close to unrelated: the two methods that drove the validation score down fastest, Optuna and the LLM, generalised worst on the fresh seeds.

The result that every method lost to the hand-tuned reference points to what the comparison actually isolates. Manual tuning used the per-epoch validation curve, trial and error then adjusted hyperparameters at a time. Every method here sees only an end-of-search aggregate for each configuration with this hypothesis that LLM can learn from its history. The result of Chapter 4 is therefore not that search cannot beat a baseline.

Chapter 5 then asked the question the thesis set out with, by freezing the architecture and letting the model reshape the training objective from a description of how the tracked person moves. Its answer is neither of the two clean ones. Motion information does reach the decisions: withholding the motion summary removes the model's whole advantage over plain mean-squared error on radar, and the arms shown the profile choose visibly different loss settings from those that are not. But it is not established that the model uses that information better than the cheap alternatives, since a deterministic controller reading the same profile and searching not at all does better, and random search over the same six levers is statistically indistinguishable from the model. The effect does not generalise: on the infrared recording both profile-reading arms fall behind plain training, and random search is the only method that still helps. A controlled sweep of the smoothness lever alone (Section 5.2.5) locates part of the reason in the objective rather than in the search: as implemented, the prior is minimised at the constant-velocity continuation rather than at the true position, and on the recording with the sharper turns its cost falls exactly in the turning samples.

6.1 Limitations

Several boundaries should be kept in mind. The study uses two recordings from two sensors (radar throughout, and infrared for the motion experiment of Chapter 5) and one network architecture, so the conclusions are about this problem rather than hyperparameter optimization in general. The seeds measure training randomness, not population variety. They are not 30 independent people, rooms, trajectories or sensor datasets. Across the thesis as a whole the radar test set was evaluated repeatedly during the work and its results were read between experiments, so it is a fixed held-out block rather than an untouched external test set, and development decisions were not made blind to it. Broader generalization would require independent trajectories, subjects, rooms or recordings. All LLMs are open models in the 4–14B range served locally; a frontier-scale model may behave differently, though the absence of any trend across the roughly threefold size range tested here offers little reason to expect it. Finally, the strict validator rejects, not repairs malformed proposals, so part of the LLM's budget is lost to a design choice that a more permissive pipeline would recover, at the cost of no longer comparing like with like.

6.2 Future Work

The most direct continuation follows from a defect in the objective rather than in the search. The smoothness penalty of Chapter 5 is minimised not at the true position but at the constant-velocity extrapolation of the two before it, so at the weights the grid offers its optimum lies between 88 and 98 per cent of the way toward that extrapolation on radar, and between 73 and 94 per cent on infrared. The term is positive at the correct answer, and fines the model for the turns the person actually made. Making the acceleration penalty a hinge, active only above a physical ceiling, would restore a zero floor and remove the bias toward straight-line motion; because the term in its present

form acts as a generic smoother rather than as a statement about human movement, that change is also available as a test of the interpretation offered in Section 5.1.2.

A second experiment follows from the infrared result. Every arm showed the motion profile set the speed ceiling at the lowest value in the grid, half a metre per second, in all searches, although the recorded trajectory reaches 5.56 m/s; the only arm to improve on baseline was the random search, which switched the velocity penalty off in nine searches. That evidence comes from the settings a joint six-lever search happened to select, so it identifies a mechanism without establishing it. Holding the other five levers neutral and sweeping the velocity penalty alone, on both recordings, would establish it or rule it out.

Three smaller questions remain open on the search side. Whether the model can read the payload at all has not been measured separately from whether it optimizes well: presenting a fixed history and asking the model to name the best and worst settings it contains, and which values proved useful, would score comprehension directly and requires no training at all. Whether selection is the bottleneck can be tested by repeating the protocol against several validation splits instead of the single 400-frame block used throughout, which Section 4.3.6 identifies as a candidate explanation for configurations that improve on validation and lose on test. And the fixed reference and the rule-based controller each contribute a single selected configuration to every comparison, so the tables carry no across-search uncertainty on the side the LLM is most often measured against; varying the seeds that drive those two arms would supply it.

Appendix A

Manual Tuning of the Baseline Configuration

Table A.1. Manual tuning trajectory of the reference configuration. Each row is a single deliberate change from the row above, selected based on validation error and on the shape of the training curve. Configurations are given as (learning rate, weight decay, dropout, window, hidden units, layers, batch size, patience); (—) means early stopping was not in use at that step. All trials trained on seeds 42 and 329 696, except the final stability check, which used 30 seeds. The test set was never inspected during tuning.

#	Configuration	Change	Val. RMSE	Curve observation
Split 70/15/15				
1	1e-2, 0, 0, 5, 32, 1, 32, 5	start	0.451	train and val plateau early, early stopped
2	1e-2, 0, 0, 10, 32, 1, 32, 10	window 5 → 10: more context; patience 5 → 10	0.395	train improved, val still high, early stopped
3	1e-2, 0, 0, 10, 64, 1, 32, 10	hidden 32 → 64: capacity	0.362	better, val begins to diverge at end
4	1e-2, 0, 0, 10, 64, 2, 32, —	layers 1 → 2: depth; window 10 → 15; drop patience	0.321	overfit: train and val separate after epoch 15
5	1e-2, 1e-4, 0, 15, 64, 2, 32, —	weight decay 0 →1e-4: L2	0.315	divergence constrained, still overfitting
Split 80/10/10				
6	1e-2, 1e-4, 0.2, 15, 64, 2, 32, —	dropout 0 → 0.2	0.298	overfit delayed, val curve noisy
7	1e-2, 1e-4, 0.2, 15, 64, 2, 64, —	batch 32 → 64: de-noise	0.285	smoother
8	1e-2, 1e-4, 0.2, 15, 64, 2, 64, —	window 15 → 20	0.271	deeper minimum, but oscillates at the end
9	5e-3, 1e-4, 0.2, 20, 64, 2, 64, —	learning rate 1e-2 → 5e-3: settle	0.259	stable descent, slow late overfit
10	5e-3, 1e-3, 0.2, 20, 64, 2, 64, —	weight decay 1e-4 → 1e-3	0.252	good generalization
Split 60/20/20				
11	5e-3, 1e-3, 0.2, 12, 64, 2, 64, 20	window 20 → 12, add patience 20	0.246	consistent gain; descent still aggressive
12	1e-3, 1e-3, 0.2, 12, 64, 2, 64, 20	learning rate 5e-3 → 1e-3	0.239	very smooth; stopped while still dropping
13	1e-3, 1e-3, 0.2, 12, 64, 2, 64, 12	patience 20 → 12	0.236	stable, small train/val gap
14	1e-3, 1e-3, 0.3, 12, 64, 2, 64, 12	dropout 0.2 → 0.3 *(selected)*	0.2308	tight, stable curves
15	1e-3, 1e-3, 0.3, 12, 64, 2, 64, 12	final configuration (stability check)	0.2308	smooth across all seeds; robust std

Table A.2. Each manual change of Table A.1 set against the rule-based controller of Section 3.5.3. Observation records whether the controller receives the signal that triggered the human's change, Rule whether it has a rule keyed to that signal, and Same whether it would propose the same move ((dir.) means the same direction on the same hyperparameter, but not the same step). Of the thirteen changes, four are moves the controller could mirror; the rest were driven by per-epoch curve dynamics it never receives, or require steps outside its move set.

Manual move	Human trigger (observed)	Obs.	Rule	Same	Gap
window 5 → 10 (+ patience 5 → 10)	underpowered; early stopped too soon	partly	yes	dir.	grid floor is 10, cannot do 5 → 10; early-stop timing invisible
hidden 32 → 64	val still high, needs capacity	no	arch-on	maybe	no capacity prior; architecture frozen by default
layers 1 → 2 (+ window 10 → 15)	val diverges at the end	no	weak	no	keys on a curve event; multi-parameter move
weight decay 0 →1e-4	still overfitting after epoch 15	partly	yes	dir.	tries dropout first; epoch timing invisible
dropout 0 → 0.2	fight overfit	yes	yes	yes	clean match
batch 32 → 64	val curve noisy	no	no	no	curve noise invisible
window 15 → 20	smoother, add temporal context	no	weak	maybe	no dataset or context prior
learning rate 1e-2 → 5e-3	oscillates at the end	no	weak	no	oscillation ≠ cross-seed variation
weight decay 1e-4 → 1e-3	slow late overfit creep	partly	yes	dir.	keys on a trend it cannot see
window 20 → 12 (+ patience 20)	switch to the protocol split	no	no	no	external protocol decision, not metric-driven
learning rate 5e-3 → 1e-3	descent too aggressive	no	yes	maybe	per-epoch slope invisible
patience 20 → 12	set protocol patience; stable	partly	yes	maybe	protocol value, not metric-triggered
dropout 0.2 → 0.3	close small train/val gap	yes	yes	yes	match

Appendix B

The Localizer Network

The network described in Section 4.1.2 is listed below as it is implemented. Two details are easier to see here than in prose: the recurrent state returned by the LSTM is discarded rather than carried between windows, and the `dropout` argument is passed to `nn.LSTM` only when more than one layer is present, since a single-layer network has no inter-layer connection on which to apply it.

Listing B.1. The `LSTM_Localizer` class (`pipeline/model.py`). Imports and the module docstring are omitted.

```
class LSTM_Localizer(nn.Module):

    def __init__(
        self,
        input_features: int,
        target_dim: int,
        lstm_hidden: int = 128,
        lstm_layers: int = 1,
        dropout: float = 0.1,
    ):
        super().__init__()
        self.lstm = nn.LSTM(
            input_features, lstm_hidden, num_layers=lstm_layers,
            batch_first=True, dropout=dropout if lstm_layers > 1 else 0.0,
        )
        self.head = nn.Sequential(
            nn.Linear(lstm_hidden, 64), nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(64, target_dim),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        out, _ = self.lstm(x)
        return self.head(out[:, -1])

    def count_parameters(self) -> Tuple[int, int]:
        total = sum(p.numel() for p in self.parameters())
        trainable = sum(p.numel() for p in self.parameters() if p.requires_grad)
        return total, trainable
```

Appendix C

The Agent: Prompts, Inputs and Outputs

This appendix reproduces the agent of Section 3.3 as it runs. Everything below is transcribed: the system prompts are the strings the code emits, and the two exchanges are taken from the stored transcripts of runs reported here, identified in their captions. Every proposer call writes its system prompt, its user message including the retry variant, the raw reply and a full-fidelity numeric snapshot of the history to the run's protocol log, so any attempt of any run can be recovered this way and none of it has to be inferred from the source.

Three notational points apply throughout. The listings are set in a monospaced font at the width of the text block, and lines longer than that width are wrapped rather than truncated. The original text contains a small number of non-ASCII characters, which are rendered here as their ASCII equivalents: an em dash as `--`, and the arrows of the trend column as `->`, `v` for improved and `^` for worsened. Finally, the hyperparameter grid printed inside the system prompt contains `window_size=12`, which is the baseline window injected into the shared grid for the radar recording as described in Section 3.2; running on another recording changes that one entry and nothing else.

C.1 The system prompt

The system message is fixed for the whole run. It is identical on the first and the twenty-fifth attempt, and it is the only place the search space is stated.

Listing C.1. The system prompt of the hyperparameter agent, as emitted by `protocol_system_prompt()` for the radar recording with architectural changes permitted.

```
You are tuning an LSTM model for indoor localization of one person. The model receives
    windows of sensor features and predicts position. Your goal is to reduce validation
    error while avoiding unstable behavior. Do not repeat a setting that has already been
     tried.

You may change ONLY these hyperparameters, and ONLY to one of the listed allowed values:
  - learning_rate: 1e-05, 3e-05, 0.0001, 0.0003, 0.001, 0.003
  - weight_decay: 0, 1e-06, 1e-05, 0.0001, 0.001
```

```
  - dropout: 0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5
  - batch_size: 32, 64, 128
  - lstm_hidden: 64, 128, 256
  - lstm_layers: 1, 2, 3
  - window_size: 10, 12, 20, 30, 40, 50
  - optimizer_choice: adam, adamw
  - patience: 8, 12, 16

Each setting is trained 3 times (3 fixed seeds) and scored by its MEAN validation RMSE -
    lower is better. You are shown the best settings so far, the most recent attempts,
    every setting already tried, and observed patterns, all as qualitative summaries.

Propose a SMALL change relative to the best setting so far (shown as ANCHOR). Respond
    using EXACTLY these lines, one field per line, no prose, no markdown:

diagnosis: <healthy | possible overfitting tendency | possible underfitting tendency |
    plateau | unstable | inconclusive>
strategy: <one short phrase, e.g. increase regularization>
changes: <param=value, param=value>
reason: <one sentence>
confidence: <low | medium | high>
```

Two sentences of that prompt are not constant across the thesis, because they describe the history block and must therefore change whenever the history block does. The sentence beginning Each setting is trained 3 times is rewritten when the payload carries measured numbers, numbers with their uncertainty, or ranks alone, and the sentence beginning propose a small change is replaced when the anchor is withheld, and a complete setting is required instead. The variant reported in this thesis is the one that replaces the labels with the measured scores, in Section 4.3.3; the others are implemented and selectable but are not among the runs presented here. In every case the prompt and the payload are generated from the same switches, for one reason: a prompt that promised information the payload did not carry would confound every measurement made with it.

C.2 The user message

The user message is rebuilt at every attempt. Listing C.2 is a complete one, taken from the fifth attempt of a real search, that is, after four settings had been trained and scored.

Listing C.2. The user message of attempt 5, transcribed from the protocol log of seed 42 of the `history-none-gemma4` run. Four settings have been evaluated, so all five blocks are populated except the pattern miner, which needs at least four scored attempts before it will speak.

```
== ANCHOR (best setting so far - propose changes relative to this) ==
  learning_rate=0.001, weight_decay=0.0001, dropout=0.25, batch_size=32, lstm_hidden=128,
       lstm_layers=2, window_size=30, optimizer_choice=adamw, patience=16

== BEST SETTINGS SO FAR (ranked #1 = best; level 1/10 = best) ==
  #1 [attempt 3] (level 1/10) learning_rate=0.001, weight_decay=0.0001, dropout=0.25,
      batch_size=32, lstm_hidden=128, lstm_layers=2, window_size=30, optimizer_choice=
      adamw, patience=16
    validation quality: best (level 1/10, 1=best 10=worst)
```

```
    reliability across 3 trainings: low
    train/validation gap: small
    best epoch timing: early
    behavior label: healthy
  #2 [attempt 1] (level 3/10) learning_rate=0.001, weight_decay=0.0001, dropout=0.3,
      batch_size=64, lstm_hidden=128, lstm_layers=2, window_size=30, optimizer_choice=
      adamw, patience=16
    validation quality: good (level 3/10, 1=best 10=worst)
    reliability across 3 trainings: low
    train/validation gap: medium
    best epoch timing: early
    behavior label: healthy
  #3 [attempt 2] (level 6/10) learning_rate=0.001, weight_decay=0.0001, dropout=0.35,
      batch_size=128, lstm_hidden=128, lstm_layers=2, window_size=30, optimizer_choice=
      adamw, patience=16
    validation quality: average (level 6/10, 1=best 10=worst)
    reliability across 3 trainings: low
    train/validation gap: medium
    best epoch timing: early
    behavior label: healthy
  #4 [attempt 4] (level 10/10) learning_rate=0.001, weight_decay=0.0001, dropout=0.25,
      batch_size=32, lstm_hidden=256, lstm_layers=2, window_size=30, optimizer_choice=
      adamw, patience=16
    validation quality: poor (level 10/10, 1=best 10=worst)
    reliability across 3 trainings: low
    train/validation gap: medium
    best epoch timing: early
    behavior label: inconclusive

== LAST ATTEMPTS (oldest -> newest; watch the level trend, 1=best 10=worst) ==
  [attempt 1] level 3/10 (good) | changed: patience=16, optimizer_choice=adamw,
      weight_decay=0.0001, lstm_hidden=128, dropout=0.3, batch_size=64, lstm_layers=2,
      window_size=30, learning_rate=0.001
    reliability: low gap: medium best epoch: early
    behavior label: healthy
    output: clean
  [attempt 2] level 6/10 (average) ^ worsened | changed: dropout=0.35, batch_size=128
    reliability: low gap: medium best epoch: early
    behavior label: healthy
    output: clean
  [attempt 3] level 1/10 (best) v improved | changed: dropout=0.25, batch_size=32
    reliability: low gap: small best epoch: early
    behavior label: healthy
    output: clean
  [attempt 4] level 10/10 (poor) ^ worsened | changed: lstm_hidden=256
    reliability: low gap: medium best epoch: early
    behavior label: inconclusive
    output: clean

== ALREADY TRIED (do NOT repeat) ==
  #1: learning_rate=0.001, weight_decay=0.0001, dropout=0.3, batch_size=64, lstm_hidden
      =128, lstm_layers=2, window_size=30, optimizer_choice=adamw, patience=16
  #2: learning_rate=0.001, weight_decay=0.0001, dropout=0.35, batch_size=128, lstm_hidden
      =128, lstm_layers=2, window_size=30, optimizer_choice=adamw, patience=16
```

```
  #3: learning_rate=0.001, weight_decay=0.0001, dropout=0.25, batch_size=32, lstm_hidden
      =128, lstm_layers=2, window_size=30, optimizer_choice=adamw, patience=16
  #4: learning_rate=0.001, weight_decay=0.0001, dropout=0.25, batch_size=32, lstm_hidden
      =256, lstm_layers=2, window_size=30, optimizer_choice=adamw, patience=16

== OBSERVED PATTERNS ==
  (not enough data yet)

Respond ONLY in the protocol line format. One field per line. No prose, no markdown.
```

The reply that message produced is Listing C.3. It is a well-formed proposal: one hyperparameter, an allowed value, and a configuration not in the tried list, so it was accepted on the first call.

Listing C.3. The model's reply to Listing C.2 (`gemma4-12B`), accepted without a retry.

```
diagnosis: healthy
strategy: increase window size
changes: window_size=40
reason: Increasing the window size to 40 may provide more temporal context for movement
    patterns while maintaining the stability of the current architecture.
confidence: medium
```

Every number in that payload is a label. Table C.1 gives the rules that produced them. The rule-based controller of Appendix D reads the same labels, computed by the same functions, which is what makes the two arms differ only in what they do with a label once they have it.

C.3 Validation, feedback and the retry

A parsed reply is a delta, not a configuration. The validator applies the four checks of Section 3.4 and returns either the merged configuration or one of the reasons in Table C.2. On a rejection, the sentence in the third column is appended to the unchanged user message, under the line

> `YOUR PREVIOUS REPLY WAS REJECTED:` (sentence)
> `Reply again, in the exact format, with a different valid change.`

and the model is called once more with that extended message. The system message, the anchor and the history are untouched, so the only thing the second call has that the first did not is the diagnosis of its own failure. A second failure ends the attempt.

C.4 The prompt after motion knowledge

Listing C.4 is the system prompt of the motion agent of Chapter 5. Set against Listing C.1, the two differences of substance are the search space, six loss-shaping levers in place of nine hyperparameters, and the paragraph explaining what each lever does in physical terms, which is the only part of either prompt that appeals to knowledge outside the search itself. The reply template is unchanged.

Table C.1. The rules that convert a measurement into the label the agent is shown. The score of a setting is its mean validation RMSE over the three trainings, its deviation is the standard deviation over the same three, and all attempts is every scored attempt of the current search. The behavior label is read top to bottom and stops at the first row that matches.

Label	Computed from	Thresholds
validation quality	this score against every scored attempt	*best* if within 2% of the lowest score seen; otherwise by the fraction f of scored attempts strictly better than it: *good* if $f < 0.34$, *average* if $f < 0.67$, *poor* otherwise
level 1–10	this score, scaled between the best and the worst seen	$1 + 9(s - s_{\min})/(s_{\max} - s_{\min})$, rounded; 1 is best
reliability across 3 trainings	coefficient of variation, deviation over score	*low* if < 0.05, *medium* if < 0.15, *high* otherwise
train/validation gap	the gap divided by the validation loss	*small* if < 0.10, *medium* if < 0.30, *large* otherwise
best epoch timing	best epoch over the 100-epoch budget	*early* if < 0.33, *middle* if < 0.66, *late* otherwise
behavior label	the four labels above	unstable if reliability is high; else possible overfitting tendency if the gap is large; else possible underfitting tendency if the gap is small and quality is poor; else healthy if quality is best, good or average; else inconclusive
observed patterns	each hyperparameter value tried at least twice, once four attempts have been scored	reported as poor or unstable if every attempt using it was so labelled, as strong if every one was best or good, and not reported otherwise; at most eight lines

Listing C.4. The system prompt of the motion agent, as emitted by `motion_loss_system_prompt()`.

```
You are improving the TRAINING OBJECTIVE of a fixed LSTM that localizes one walking
    person indoors. The network architecture and all conventional hyperparameters are
    FROZEN -- you change ONLY the loss function, using your knowledge of how humans move.

You are given interpretable MOTION SUMMARIES of the tracked person (walking speed,
    acceleration, turning, stop-go/dwell, trajectory roughness) and, once settings have
    been tried, where the model's position error concentrates across motion regimes (slow
     / medium / fast).

You set six loss-shaping levers, ONLY to one of the listed allowed values:
  - v_max: 0.5, 0.75, 1, 1.5, 2
  - lambda_vel: 0, 0.05, 0.1, 0.2, 0.3
  - lambda_smooth: 0, 0.05, 0.1, 0.2, 0.3
```

Table C.2. Every way a proposal can fail, the tag recorded in the log, and the sentence fed back to the model. The first two are failures to obtain a reply, the third a reply with no recoverable fields, and the rest failures of the proposal itself; all of them consume the attempt if they recur on the retry. Incomplete setting can arise only in the payload variant that withholds the anchor, since only there is a complete configuration demanded; it does not occur in the runs reported here. A reply that parses but names no hyperparameter has no row of its own: its empty delta reproduces the anchor and is caught by already tried.

Failure	Logged tag	Sentence returned to the model
The call did not return	`llm_call_timeout`	the previous call did not return a usable reply; reply concisely in the exact format
The server errored	`client_error`	as above
No five fields found	`parse_error`	your reply could not be parsed; output exactly the 5 fields, one per line
Not a hyperparameter	`unknown_param:`*k*	*k* is not a tunable parameter
Architecture is frozen	`arch_frozen:`*k*	*k* is fixed this run; do not change it
Value off the grid	`value_not_in_grid:` *k*=*v*	*k*=*v* is not an allowed value; pick one from the list
Configuration repeated	`already_tried`	that setting was already tried; propose a different change
Incomplete setting	`incomplete_setting:` k_1,k_2	give a COMPLETE setting – these are missing: k_1,k_2

```
  - bin_weight_slow: 0.5, 1, 1.5, 2, 3
  - bin_weight_medium: 0.5, 1, 1.5, 2, 3
  - bin_weight_fast: 0.5, 1, 1.5, 2, 3

What the levers do (this is where human-motion knowledge matters):
  - v_max -- plausible top human walking speed in m/s. Predicted steps faster than this
      are penalised. Set it just above the observed p95 speed (about 1.1x); do not leave
      a generic default if the data's speed range differs.
  - lambda_vel -- strength of that speed-plausibility penalty (0 = off). Raise it when
      predictions look noisy or the motion is smooth/slow.
  - lambda_smooth -- penalty on physically implausible acceleration (0 = off). Its scale
      is relative to this person's own typical acceleration: 0.05 is a light nudge, 0.3
      weighs about as much as the position error itself. Raise it when the trajectory is
      smooth and dwell episodes are frequent; keep low when motion is genuinely jerky/
      fast.
  - bin_weight_slow / medium / fast -- per-speed-regime position-error weights (1.0 =
      neutral). Up-weight whichever regime the model fits worst (use the per-regime error
      ), e.g. the fast regime when quick movements are hardest.

Neutral levers (lambda_vel=0, lambda_smooth=0, all bin weights 1.0) reduce to plain MSE.
    Propose a COMPLETE lever vector (all six) that you expect to lower validation error.
    Do not repeat a lever vector already tried.

Respond using EXACTLY these lines, one field per line, no prose, no markdown:
```

```
diagnosis: <healthy | possible overfitting tendency | possible underfitting tendency |
    plateau | unstable | inconclusive>
strategy: <one short phrase, e.g. penalise implausible speed, upweight fast regime>
changes: v_max=<val>, lambda_vel=<val>, lambda_smooth=<val>, bin_weight_slow=<val>,
    bin_weight_medium=<val>, bin_weight_fast=<val>
reason: <one sentence grounded in the motion summaries>
confidence: <low | medium | high>
```

The user message changes with it. Listing C.5 is a real one, and the contrast with Listing C.2 is visible at a glance: the motion summary is given as measured numbers in SI units, and so are the scores of the lever vectors already tried. That reversal is deliberate rather than an inconsistency. In the hyperparameter prompt, a number is a measurement of a training run, and it is withheld to test whether the model can act on the trend without it; here a number is a physical fact about a walking person, 0.60 m/s at the 95th percentile, and it is shown precisely because the knowledge under test is quantitative. The prompt asks for a speed ceiling a little above that percentile, which cannot be expressed as a label. Note also the second line of the lever block, which reports where the current best setting's error falls across the three speed regimes; it is the feedback signal the motion agent has, and the hyperparameter agent does not.

Listing C.5. The user message of a motion attempt, transcribed from the motion protocol log of seed 42 of the `motion-qwenv3` run, with the reply it produced. The motion summary is that of the radar training split, the same numbers plotted in Figure 5.1.

```
== MOTION SUMMARY of the tracked person (raw numbers) ==
  speed m/s: mean 0.302, median 0.284, p95 0.6, max 3.34, std 0.191
  acceleration m/s^2: mean 0.592, p95 1.87
  turning: mean 26.5 deg/step, p95 114 deg, sharp-turn share 0.201
  roughness: path sinuosity 37.6 (1=straight), mean jerk 4.17 m/s^3
  stop-go: stop share 0.03, 6.6 dwell episodes/min

== CURRENT LOSS-SHAPING LEVERS (propose changes relative to this) ==
  v_max=0.75, lambda_vel=0.05, lambda_smooth=0.05, bin_weight_slow=1, bin_weight_medium
      =1, bin_weight_fast=2
  best-so-far error by regime (metres): slow 0.2478, medium 0.2639, fast 0.2614 -> worst
      = medium (spread x1.06)

== TRIED LEVER VECTORS (score = mean validation RMSE in m, lower better; worst -> best)
    ==
  score=0.2202 v_max=1, lambda_vel=0.1, lambda_smooth=0.05, bin_weight_slow=1,
      bin_weight_medium=1.5, bin_weight_fast=2
  score=0.2155 v_max=0.75, lambda_vel=0.05, lambda_smooth=0.1, bin_weight_slow=1,
      bin_weight_medium=2, bin_weight_fast=2
  score=0.212 v_max=0.75, lambda_vel=0.05, lambda_smooth=0.05, bin_weight_slow=1,
      bin_weight_medium=1, bin_weight_fast=2

Propose ONE complete lever vector (all six), different from every vector above, that
    lowers validation error.
Respond ONLY in the protocol line format. One field per line. No prose, no markdown.

--- the reply ---

diagnosis: possible underfitting tendency
strategy: upweight medium regime, increase speed plausibility penalty
```

```
changes: v_max=0.75, lambda_vel=0.1, lambda_smooth=0.05, bin_weight_slow=1,
    bin_weight_medium=2, bin_weight_fast=2
reason: model's worst error is in medium-speed regime (0.2639), requiring stronger focus
    on this regime and tighter speed constraints to reduce implausible predictions
confidence: medium
```

That reply is worth one remark, because it is the behavior Section 5.2.7 quantifies. The `reason` line does cite the per-regime figure it was given, and the proposed move follows from it. What it does not do is use the motion summary above it: the recommended recipe would set the speed ceiling to about 1.1 times the 95th-percentile speed of 0.6 m/s, and the reply leaves that lever where it found it.

Appendix D

The Rule-Based Controller

The controller of Section 3.5.3 is the arm that answers what the language model is being paid for. It receives the same history, reads it into the same six diagnosis labels by the same functions, and is confined to the same grid; the only thing it lacks is the reasoning step. This appendix gives it in full, so that the comparison can be checked.

It proposes in two stages. It first assigns a diagnosis to the most recent attempt, using nothing but the labels of Table C.1; this is the same computation whose output appears as `behavior label` in Listing C.2. It then walks that diagnosis's list of candidate moves, given in Table D.1, and returns the first one that is admissible.

Listing D.1. The controller's proposal step. The diagnosis stage is `_diagnose_protocol` and the action stage `_act_protocol` in `arms/rule_based.py`.

```
INPUT anchor the best configuration so far
       history every attempt so far, with its labels
       is_tried(c) true if configuration c has been evaluated
       arch_allowed whether lstm_hidden / lstm_layers may move
OUTPUT a single-hyperparameter delta against the anchor, or a rejection

# ---- stage 1: name the situation -----------------------------------
function DIAGNOSE(history):
    if history is empty: return inconclusive
    last <- history[-1]
    if last was not trained: return inconclusive

    # curve-aware variant only (the per-epoch curve-summary experiment):
    # prefer the curve shape when it is decisive, which is the one extra
    # signal that arm is given. The default controller skips this block.
    if curves_enabled:
        shape <- CURVE_SHAPE(last)
        if shape is still_improving: return possible_underfitting
        if shape is overfitting_upturn: return possible_overfitting
        if shape is unstable_noisy: return unstable
        # clean_plateau adds nothing here, so fall through

    # otherwise the aggregate labels of the label-threshold table, in order
    variation <- RELIABILITY(last) # low / medium / high
    gap <- GAP(last) # small / medium / large
    quality <- QUALITY(last, history) # best / good / average / poor
```

```
    if variation is high: return unstable
    if gap is large: return possible_overfitting
    if gap is small and quality is poor: return possible_underfitting
    if quality in {best, good, average}: return healthy
    return inconclusive

# ---- stage 2: act on it ------------------------------------------------
function ACT(diagnosis, anchor):
    # first attempt of the run: evaluate the anchor itself, so that the
    # search has one scored point to diagnose from
    if diagnosis is inconclusive and not is_tried(anchor):
        return {} , "exploit"

    moves <- MOVES[diagnosis] # move table below, in order
    if arch_allowed:
        moves <- moves + ARCH_MOVES[diagnosis] # capacity first when
                                               # underfitting, last otherwise
    for (parameter, direction) in moves:
        current <- anchor[parameter]
        next <- GRID_NEIGHBOUR(parameter, current, direction)
        if next == current: continue # at the end of the grid
        candidate <- anchor with parameter set to next
        if is_tried(candidate): continue # would repeat an evaluation
        return {parameter: next} , STRATEGY[diagnosis]

    return REJECT("exhausted") # no admissible move remains

# ---- the proposal ------------------------------------------------------
diagnosis <- DIAGNOSE(history)
return ACT(diagnosis, anchor)
```

Three properties of this procedure are worth stating explicitly, because each of them is a decision that could have been made otherwise.

It changes exactly one hyperparameter per attempt, and moves it exactly one position along that hyperparameter's grid. It cannot jump, and it cannot combine two changes, both of which the agent is free to do. The comparison is therefore not between equals in expressive power: the controller is the weaker proposer by construction, which is what makes it the right floor to measure against.

It never proposes a move it knows to be wasted. A move that would step off the end of a grid leaves the configuration unchanged and is skipped, and a move whose result has already been evaluated is skipped as well, since the validator of Section 3.4 would reject it. When every move on the list is one of those two, the controller declines to propose and the attempt is recorded as rejected, exactly as an invalid reply from the agent would be. This is the mechanism referred to in Section 3.5.3: an earlier version with one move per diagnosis proposed the same configuration on every attempt once its preferred move had been tried, and spent the remainder of the budget producing nothing.

Its tables were fixed in advance and never revised in the light of a result. The ordering within each list encodes a prior about what to try first, lower the learning rate before touching anything else when training is unstable, add regularization before reducing the learning rate when it is overfitting, and those priors are the whole of the domain knowledge this arm has.

Table D.1. The candidate moves for each diagnosis, in the order they are attempted. A sign is the direction of one step along that hyperparameter's grid. Architectural moves are appended only when architecture is unfrozen, and are tried first under underfitting, where capacity is the more likely cause, and last otherwise. Inconclusive has no list of its own: on the first attempt it evaluates the anchor, and thereafter it falls through to the healthy list rather than waste the attempt.

Diagnosis	Strategy	Moves, in order
unstable	stabilise	learning rate −, batch size −, weight decay +, dropout +, patience +, learning rate +
plateau	escape plateau	learning rate −, learning rate +, dropout −, weight decay −, batch size −, patience +
possible overfitting tendency	regularise	dropout +, weight decay +, batch size +, learning rate −
possible underfitting tendency	explore	*(hidden units +, layers +,)* learning rate +, dropout −, weight decay −, window +, batch size −
healthy	exploit	dropout ±, weight decay ±, learning rate − then +, batch size ±, patience +, window ± *(, hidden units ±, layers +)*

D.1 The motion controller

Chapter 5 needs a deterministic comparator for the motion agent, and uses the same controller with one addition: it sets the six loss-shaping levers from fixed formulas on the recording's motion profile, rather than by reasoning about it. It applies the recipe the motion prompt states, $v_{\max} = 1.1\, v_{95}$ snapped to the nearest allowed lever value, and pairs it with three constants, a velocity weight of 0.1, a smoothness weight of 0.05, and a fast-regime error weight of 1.5. On radar the ceiling works out at $1.1 \times 0.60 = 0.66$ m/s, which snaps to 0.75. The profile does not change between attempts, so neither do the levers: the arm sets them once and then behaves exactly as the controller above.

The pairing is what makes the chapter's two questions separable. This arm against the plain controller asks whether motion-aware loss shaping helps at all; the motion agent against this arm asks whether a language model's reading of the motion profile beats a formula applied to the same numbers.

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631, 2019.

[2] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proceedings of IEEE INFOCOM*, volume 2, pages 775–784, 2000.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

[5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020.

[7] Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(70):2079–2107, 2010.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[9] Siyi Liu, Chen Gao, and Yong Li. AgentHPO: Large language model agent for hyper-parameter optimization. In *Conference on Parsimony and Learning (CPAL)*, volume 280 of *Proceedings of Machine Learning Research*, pages 1146–1169. PMLR, 2025.

[10] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *International Conference on Learning Representations (ICLR)*, 2024.

[11] Lennart Schneider, Bernd Bischl, and Matthias Feurer. Overtuning in hyperparameter optimization. In *Proceedings of the Fourth International Conference on Automated Machine Learning (AutoML)*, volume 293 of *Proceedings of Machine Learning Research*, pages 17/1–43. PMLR, 2025.

[12] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[13] Akash Deep Singh, Sandeep Singh Sandha, Luis Garcia, and Mani Srivastava. RadHAR: Human activity recognition from point clouds generated through a millimeter-wave radar. In *Proceedings of the 3rd ACM Workshop on Millimeter-Wave Networks and Sensing Systems (mmNets)*, pages 51–56, 2019.

[14] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 5998–6008, 2017.

[16] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837, 2022.

[17] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024.

[18] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. A survey of indoor localization systems and technologies. *IEEE Communications Surveys and Tutorials*, 21(3):2568–2599, 2019.

[19] Mingkai Zheng, Xiu Su, Shan You, Fei Wang, Chen Qian, Chang Xu, and Samuel Albanie. Can GPT-4 perform neural architecture search? *arXiv preprint arXiv:2304.10970*, 2023.