# Exploring the Viability of Automated Heuristic Design for 5G LDPC Decoding

Reza Namvar, José Gallego, Albert Banchs
*IMDEA Networks Institute and Universidad Carlos III de Madrid*
Madrid, Spain
{name.surname}@networks.imdea.org

Livia Elena Chatzieleftheriou
*Delft University of Technology*
Delft, Netherlands
L.E.Chatzieleftheriou@tudelft.nl

Jose A. Ayala-Romero, Andrés Garcia-Saavedra
*NEC Laboratories Europe*
Heidelberg, Germany
{name.surname}@neclab.eu

Marco Fiore
*IMDEA Networks Institute*
Madrid, Spain
marco.fiore@networks.imdea.org

***Abstract*—Automated Heuristic Design (AHD) leverages Large Language Models (LLMs) and task-specific evaluation to search among existing algorithmic components and generate novel ones. This paper studies AHD for 5G New Radio (NR) Low-Density Parity Check (LDPC) decoding by evolving the Check Node Update (CNU) function used in iterative Belief Propagation (BP). To this end, we implement a flexible AHD framework capable of accommodating different evolution policies and assess their performance (*i.e.*, the decoding accuracy achieved by the discovered heuristic) and computational complexity (*i.e.*, the number of evaluated candidate heuristics) in the target 5G NR task. We experiment with two evolution policies: LLM-Based Evolution (LBE), which performs population-based parallel mutation and selection, and Prompt-guided LLM-Based Evolution (PLBE), which augments evolution with structured prompt operators. Under a fixed time budget per experiment, we find that AHD prompted by specifying the context of the task (*i.e.*, LDPC decoding) consistently converges toward state-of-the-art performance and is robust to the specific evaluation approach employed. Instead, context-agnostic prompting and/or exploratory parent sampling tend to stagnate at substantially lower scores. The best discovered CNU heuristic is structurally close to functions employed in production 5G networks and marginally outperforms such functions on the specific Transport Block (TB) batch used for AHD. However, the additional gain disappears under independently drawn TBs. Ultimately, our study highlights the promise of AHD for 5G tasks but also the need for careful validation when interpreting in-loop gains.**

***Index Terms*—Automatic Heuristic Design, network operation, heuristic algorithms, Large Language Models, LDPC decoding.**

## I. Introduction

LDPC codes are a cornerstone of modern wireless communication systems, and their iterative decoders remain among the most performance-critical components in the physical layer of a modern mobile network. The optimal solution for LDPC decoding requires solving a maximum-a-posteriori problem, which is computationally very complex. Hence, the problem is approximated in real system by an iterative BP heuristic so as to satisfy the tight latency and reliability constraints that characterize practical 5G NR receivers.

Improving heuristics for BP requires time-consuming expert investigations, making very valuable AHD methods that can automate parts of the design and produce deployable code.

Recent progress in AHD set forth LLM-based simulator-in-the-loop workflows, where LLMs propose executable code candidates, an external evaluator scores them under system-relevant metrics, and an evolutionary loop maintains and refines a population of candidates. This approach is particularly appealing for communication systems because it produces code-level heuristics rather than end-to-end black-box models, as well as because high-fidelity simulation pipelines already exist to provide grounded feedback. FunSearch [1] popularized distributed island-based evolution with best-shot prompting; Evolution of Heuristics (EoH) [2] co-evolves natural-language "thoughts" and code using structured prompt operators; and Reflective Evolution (ReEvo) [3] injects reflective feedback to guide iterative refinement. Recent extensions emphasize richer objectives and exploration strategies, including multiobjective population management (MEoH) [4], algorithm–model co-evolution (CALM) [5], and diversity-driven operators (HSEvo) [6].

In this paper, we study the viability of AHD for a concrete and computation-critical component of 5G NR LDPC decoding: synthesizing variants of the CNU inside iterative BP. We focus on the best discovered candidate, while constraining our search under a fixed evaluation time budget. To this end, we use the score trace as the main diagnostic, *i.e.*, the best-achieved score plotted against the number of generated programs, which directly captures sample efficiency and stagnation behavior. We compare two evolution policies built on the same evaluator and scoring pipeline: LBE, a population-based parallel evolution strategy inspired by program-search engines such as FunSearch [1], and PLBE, which incorporates structured prompt operators inspired by Evolution of Heuristics [2] to encourage systematic refinement while maintaining the same evaluation interface.

Our baselines are `boxplus` and `boxplus-phi`, which are

979-8-3315-6128-4/26/$31.00 ©2026 IEEE

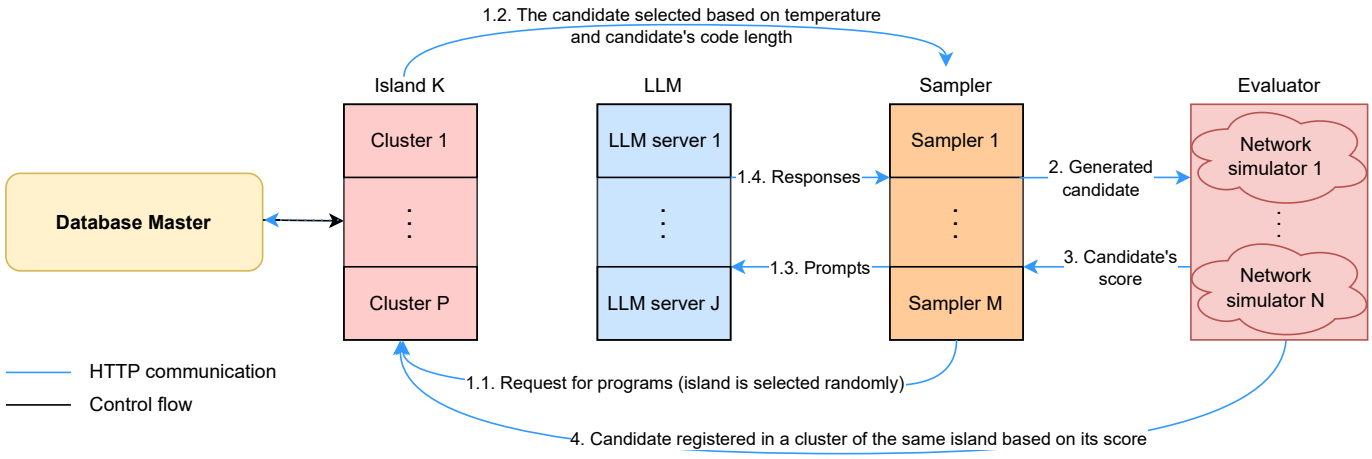



Fig. 1. General overview of AHD architecture implemented for our study

outcomes of extensive, community-driven research and engineering over decades, and our goal is to quantify what an AHD loop can discover when carefully integrated in a realistic 5G NR decoding pipeline [7]. Concretely, we run controlled ablations over (i) evaluation breadth (*i.e.*, number of configuration points used for in-loop scoring), (ii) parent-selection policy (*i.e.*, exploratory versus exploitative sampling), and (iii) prompt grounding (*i.e.*, generic versus LDPC-specific instructions), and we compare LBE and PLBE under the same evaluation infrastructure. Our results show that LDPC-grounded prompting is a key enabler of convergence toward baseline-like CNU behavior, and that near-baseline score traces are achievable across multiple evaluation breadths under the same effective recipe.

The remainder of the paper is organized as follows. Section II describes the AHD framework and its integration with the 5G NR LDPC decoder. Section III presents the experimental design and ablations. Section IV discusses score-trace results, controlled in-loop gains, and robustness under independent TB draws. Section V concludes the paper.

## II. System Framework and Methodology

We start by describing our distributed AHD framework, then present the networking problem that we aim to tackle using our framework, followed by the presentation of how the produced candidate heuristic would integrate with the LDPC decoder. We conclude with the evaluation, scoring and score-tracing instrumentation.

### A. Distributed AHD framework and evolution modes

**AHD Workflow.** We present the operation of our proposed AHD framework through a four-step cycle in Fig. 1. It begins with (1) the AHD engine leveraging LLMs to draft heuristics, followed by (2) their deployment and testing within an active network. Then, (3) performance data is routed back to the engine for evaluation and, finally, (4) the most effective heuristics are archived in a database, allowing for version tracking and the retention of high-quality code.

**Distributed, Modular Framework Architecture.** We implement our framework as a modular distributed system with separate services for coordination, population storage, sampling, LLM inference, and evaluation. Candidate programs are stored across multiple islands, where each island maintains a local population of code snippets paired with their scores. To support efficient selection, populations are organized into score-based clusters that enable sampling parents according to their performance while maintaining diversity.

**Candidate Generation and Evaluation Pipeline.** Sampler workers repeatedly request a configurable number of parent programs from islands, construct prompts by combining the parent code with an instruction, and query an external LLM server (Qwen3-30B-A3B-FP8 in our implementation) to produce a modified candidate function. The candidate is then forwarded to the evaluator, which executes it in a sandboxed environment and returns a scalar score. If the candidate produces a feasible solution it is inserted back into the population; otherwise, it is discarded. Since each component communicates via lightweight APIs, the system scales by increasing the number of islands, sampler workers, or evaluator replicas.

**Generation Policies: LBE versus PLBE.** We compare two generation policies on top of this common infrastructure. In LBE, evolution is driven by parent sampling from the population and direct LLM-based code mutation, enabling high-throughput parallel generation and evaluation of candidates. In PLBE, candidate generation uses structured prompt operators that elicit higher-level reflection and refinement before code emission, inspired by prior work on program-search and heuristic evolution. Importantly, both LBE and PLBE share the same evaluator, scoring, and insertion interface. Thus, differences can be attributed to the generation policy rather than to changes in the decoding pipeline.

### B. 5G NR LDPC decoding and the CNU synthesis target

We consider 5G NR TB decoding under iterative BP decoding of LDPC codes. At the receiver, information is

represented as Log Likelihood Ratios (LLRs), i.e. real-valued reliability metrics produced by soft demapping and subsequent rate-recovery operations. An LLR captures how strongly the receiver believes a bit is 0 or 1: the sign gives the more likely bit value (positive for 1 and negative for 0), and the magnitude indicates confidence (the higher the absolute value of the LLR, the higher the confidence). The goal of the decoder is to turn these noisy, per-bit beliefs into a codeword that satisfies all parity constraints of the LDPC code.

The decoder operates on the LDPC Tanner graph, a bipartite representation of the parity-check matrix. In this graph, "variable" nodes correspond to code bits and "check" nodes correspond to parity equations that those bits must satisfy. Decoding proceeds by repeatedly exchanging messages along the graph's edges, so that each bit's "belief" value is refined using both the channel evidence and the constraints imposed by the code.

Each iteration alternates between Variable Node Update (VNU) and CNU computations. The VNU updates the belief of a bit by combining its channel LLR with the incoming messages from neighboring check nodes, producing a new soft reliability for that bit. The CNU updates the message sent from a check node to one variable node by aggregating information from all other connected variable nodes, effectively estimating how consistent the parity equation is given the current beliefs.

After each iteration, the decoder forms hard decisions from the updated reliabilities and checks the TB Cyclic Redundancy Check (CRC) for early stopping. A passed CRC indicates a valid TB, allowing decoding to stop early and reducing latency.

Our work uses AHD targeting to improve the CNU, which is both performance-critical and structured: it must compute extrinsic messages (excluding the contribution of the recipient edge) and it is repeatedly invoked across the Tanner graph and decoding iterations. Strong hand-crafted functions such as `boxplus` and `boxplus-phi` implement this computation via stable sign management and magnitude aggregation rules. We use these well-established CNU variants as baselines and treat LLM-guided AHD as a mechanism to discover alternative CNU implementations that remain compatible with the end-to-end 5G NR decoding chain.

### C. Candidate interface and integration into the decoder

The AHD loop searches over candidate Python functions that implement the CNU as a part of LDPC 5G NR decoding pipeline in Sionna [8]. Each candidate is required to satisfy a fixed interface that matches the decoder implementation (input tensor shape conventions, dtype, and output semantics). During evaluation, the candidate CNU is injected into an otherwise unchanged link-level pipeline: the transmitter and receiver blocks, the VNU logic, the iteration schedule, and the CRC-based early termination remain fixed. This design isolates attribution, ensuring that changes in score originate from the candidate CNU and not from confounding modifications elsewhere in the chain.

Because candidate generation is an unconstrained code synthesis task, many candidates may violate interface expectations or produce unusable outputs, *i.e.*, infeasible solutions. For this reason, the evaluator treats runtime errors, timeouts, and invalid behavior as failures and discards such candidates without inserting them into the population. Only the candidate solutions that execute successfully end-to-end and return valid decoder outputs are considered feasible and are stored for future sampling.

### D. Evaluation, scoring, and score-trace instrumentation

Each candidate is evaluated at one or more operating conditions (i.e. configuration points), with each evaluation producing decoder outcomes such as the number of successfully decoded TBs, aggregate Bit Error Rate (BER), and iteration statistics. These outcomes are combined into a single scalar score used for selection. We implement this priority ordering via a weighted penalty score,

$$S \;=\; -\Big(\frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\big[P_{\text{und}}\cdot U_p \;+\; P_{\text{ber}}\cdot \text{BER}_p \;+\; P_{\text{it}}\cdot \bar{I}_p\big]\Big), \quad (1)$$

where $\mathcal{P}$ is the set of configuration points used for in-loop evaluation, $U_p$ is the number of undecoded TBs at point $p$, $\text{BER}_p$ is the post-decoding BER over decoded TBs, and $\bar{I}_p$ is the average number of decoder iterations. Larger scores, *i.e.*, less negative, are better. The score is computed only for feasible candidates that compile and execute successfully within the evaluator. Candidates that fail compilation or trigger runtime errors are assigned a fixed worst-case score of $-10^9$ and are discarded (*i.e.*, not inserted into the population). We choose penalty magnitudes to enforce a lexicographic priority among objectives, with $P_{\text{und}} = 10^7$, $P_{\text{ber}} = 10^6$, and $P_{\text{it}} = 10^0$, so that reliability (decoded TBs) dominates, followed by bit-level accuracy, and finally iteration cost.

To characterize search dynamics under a fixed time budget, we log the best-achieved score as a function of the number of generated programs, forming a score trace. This representation makes it possible to compare sample efficiency across evolution modes and ablations, and it directly reflects whether a design choice improves early progress or delays stagnation.

## III. Experimental Design

We evaluate three practical design knobs: (i) evaluation breadth using 1, 3, or 6 configuration points, (ii) exploration versus no-exploration parent selection in LBE, and (iii) prompt framing. For the latter knob, we experiment with generic prompting versus LDPC-specific prompts that explicitly instruct the LLM to improve CNU for LDPC decoding. Fig. 2–Fig. 4 show that baseline-like CNU behavior can be obtained across all three evaluation breadths when an LDPC-specific prompt grounds the search and uses no-exploration sampling; other combinations (*e.g.*, generic prompting or exploratory sampling) can stagnate at substantially lower scores.

### A. Primary metric: score trace

The main metric reported in this work is the score trace: the best-achieved score as a function of the number of generated programs. This trace captures efficiency and stagnation

behavior under a fixed evaluation budget and enables direct comparison across evolution modes and ablations.

All ablation conditions are executed under the same wall-clock budget (96 hours per run) on the same evaluation pipeline, so differences in the number of generated programs reflect throughput differences induced by evaluation breadth and generation policy.

### B. Configuration points and evaluation breadth

Candidate CNU programs are evaluated at operating conditions referred to as *configuration points*. A configuration point is a triplet (`n_prb`, `mcs_idx`, `snr_dB`) specified by the number of physical resource blocks (`n_prb`), the modulation and coding scheme index (`mcs_idx`), and the signal-to-noise ratio in dB (`snr_dB`). The parameter `n_prb` determines the allocated bandwidth and influences transport size and codeword lengths; `mcs_idx` selects the modulation order and coding rate; and `snr_dB` controls channel quality. Together, these parameters determine the effective difficulty regime of decoding, ranging from settings where most TBs decode reliably to settings where failures dominate.

A natural hypothesis is that evaluating each candidate on more configuration points provides more context to the evolutionary loop and yields a more informative selection signal. To test this, we compare the following evaluation sets, that span a mixture of easier and harder regimes while remaining computationally feasible for large-scale candidate evaluation: 1-point set: {(150, 21, 20)}. 3-point set: {(50, 4, 5), (200, 25, 25), (150, 21, 20)}. 6-point set: {(50, 4, 5), (100, 9, 12), (200, 25, 25), (150, 10, 10), (150, 21, 20), (50, 3, 1)}.

### C. Exploration versus no-exploration parent selection

Within LBE, parent selection is performed at the cluster level using a temperature-controlled sampling distribution. Let $\{s_c\}$ denote the set of current cluster scores and $T$ the global sampling temperature. Then, the probability $P(c)$ of selecting cluster $c$ is computed using the tempered softmax function:

$$P(c) = \frac{\exp(s_c/T)}{\sum_j \exp(s_j/T)}. \tag{2}$$

This mechanism allows the system to modulate selection pressure dynamically: high temperatures approximate uniform random sampling (exploration), while low temperatures approximate an $\mathrm{argmax}$ operation (exploitation).

We compare two variants for calibrating the base temperature $t_0$, differing in their sensitivity to the population spread. Let $s^\star$ denote the current best score. We instantiate:

$$\text{Exploration init:} \quad t_0 = \max_c |s_c| \cdot \alpha, \tag{3}$$

$$\text{No-exploration init:} \quad t_0 = |s^\star| \cdot \alpha, \tag{4}$$

where $\alpha$ is a fixed scaling constant. The *exploration* variant derives $t_0$ from the maximum absolute score in the population. In early stages, when low-quality programs yield large negative scores, this strategy results in a high $t_0$, effectively flattening the selection probabilities to preserve diversity. Conversely, the *no-exploration* variant scales $t_0$ solely by the magnitude of the best score, resulting in lower temperatures that concentrate sampling mass around the current best clusters.

In *exploration* mode, we employ a periodic annealing schedule. The effective temperature $T$ is linearly decayed from $t_0$ to 0 over a period of $N$ generated programs (one selection window) and then reset. The temperature as of the $n$-th generated program is given by:

$$T(n) = t_0 \cdot \left(1 - \frac{n \pmod N}{N}\right), \tag{5}$$

where $N = 2000$ in our experiments. This profile forces the search to alternate cyclically between exploring the landscape and converging on high-performing candidates. Fig. 6 and Fig. 5 visualize the resulting cluster-selection probabilities within a representative window for the no-exploration and exploration initializations, respectively.

### D. Prompt framing: generic versus LDPC-grounded

In case of LBE we compare two instruction framings used in the prompt sent to the LLM. The generic version does not mention the domain:

> "Improve the funsearch.evolve decorated function here. Try to come up with a new algorithm."

The LDPC-grounded version explicitly targets the intended component:

> "Improve the check node update function (cn_update here) for LDPC decoding. Try to come up with a new algorithm."

This ablation tests whether minimal task grounding increases the rate of viable candidates and improves the score trace.

Inspired by [2], our prompt in PLBE tries to give a high level view of the problem and is:

> "I need help designing a new heuristic for a constraint agent that computes messages for its connected components. In each step, the agent generates an outgoing message for a single target component. This message is calculated by applying a specific update rule to all the incoming messages the agent has received from its other connected components. The final goal is to repeat this process until the entire system of components reaches a stable state that satisfies all agent constraints."

We compare LBE against an PLBE implementation using the same link-level evaluator and the same scoring function. For each condition, we report the score trace (best-achieved score vs. number of generated programs).

## IV. Results and Discussion

### A. Score-trace behavior across ablations

Across the majority of LBE settings, score traces exhibit a common pattern: an initial improvement phase in which the search quickly accumulates candidates that provide valid solutions, followed by saturation where additional program generation yields little improvement in the best-achieved score. This behavior persists across evaluation breadth and exploration/no-exploration sampling variants.

In contrast, PLBE exhibits little to no improvement in the score trace in our experiments, with runs often stagnating

near their initial performance. A plausible explanation is that, under our current configuration, PLBE explores a narrower region of the search space and is therefore more prone to early convergence to suboptimal candidate families.

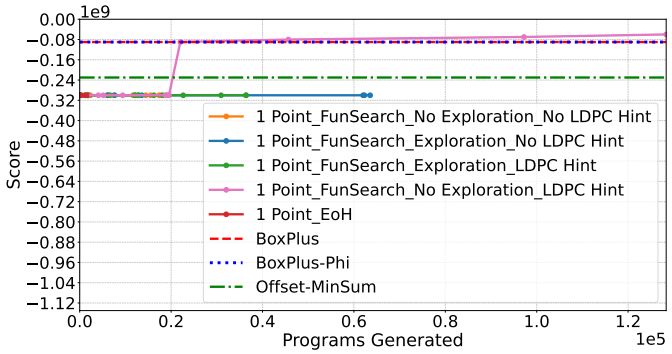



Fig. 2. Results for 1-point experiments

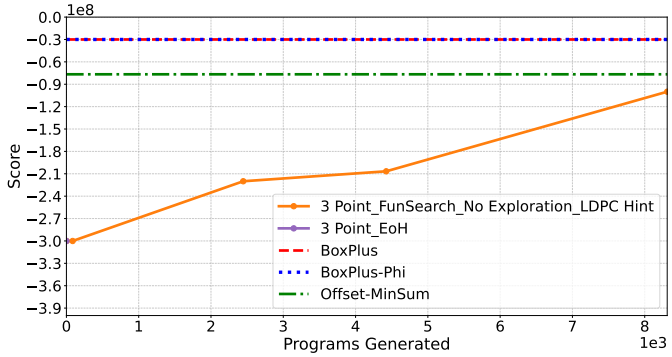



Fig. 3. Results for 3-point experiments

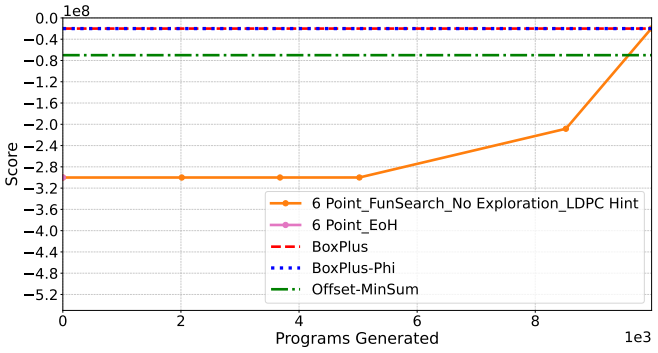



Fig. 4. Results for 6-point experiments

Fig. 2 through Fig. 4 show that evaluation breadth does not preclude reaching baseline-like scores: under no-exploration sampling and an LDPC-grounded prompt, the score trace consistently progresses toward the `boxplus` and `boxplus-phi` baselines for 1-, 3-, and 6-point evaluation. Conversely, Fig. 2 shows that generic prompting and/or exploratory sampling can stagnate far from the baselines, closer to offset-minsum-level performance. The different behaviors of exploratory and exploitative parent selection are clarified by Fig. 5–Fig. 6. With the exploration initialization, probability mass remains spread

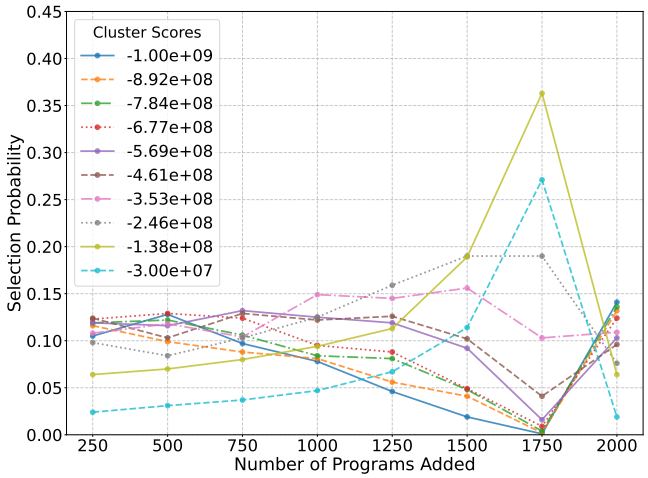



Fig. 5. Probabilities assigned to clusters in exploration mode

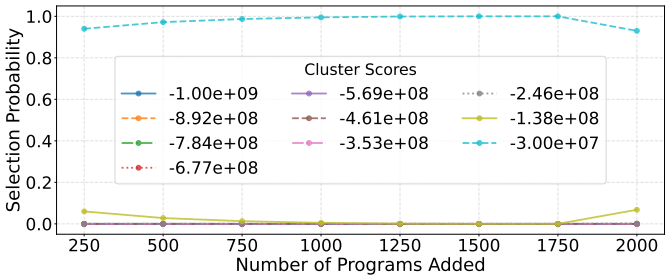



Fig. 6. Probabilities assigned to clusters in no exploration mode

across a larger set of clusters for longer (Fig. 5), maintaining diversity but also allocating substantial budget to weaker clusters. With the no-exploration initialization, probability mass collapses rapidly onto a small number of top-scoring clusters within each 2000-program window (Fig. 6), effectively concentrating sampling on the currently strongest code families. Under a fixed time budget, this trade-off can delay exploitation of high-quality regions of the search space, which is consistent with the stagnation observed in the exploratory setting in Fig. 2. Overall, these results suggest that prompt grounding and the parent-selection policy are primary determinants of successful convergence in our setting, while the number of configuration points has a secondary effect on the trace shape under a fixed program-generation budget. We also note that "number of generated programs" is not compute-normalized across 1/3/6-point evaluations, since each additional configuration point increases per-program evaluation cost.

### *B. Best-performing setting and discovered CNU quality*

Among all tested combinations, the best-performing discovered CNU is produced by LBE implementation under single-point evaluation, no-exploration sampling, and the LDPC-grounded prompt. At the same time, Fig. 3 and Fig. 4 show that near-baseline behavior is also achievable under 3-point and 6-point evaluation when using the same no-exploration and LDPC-grounded recipe, indicating that success is not unique to a single evaluation breadth. Under this setting,

the search identifies a candidate function whose structure and observed behavior are very close to the `boxplus` and `boxplus-phi` baselines. Qualitatively, the discovered update follows the same high-level organization as the established baselines, suggesting that the LLM-guided evolutionary loop can reach baseline-like algorithmic structure for a complex decoder component when the prompt explicitly anchors the target role of the function.

### C. Controlled in-loop gain and robustness evaluation

At the reference configuration point, the best discovered candidate decodes two additional TBs compared to `boxplus` and `boxplus-phi` when evaluated on the same fixed TB set used during the evolutionary evaluation loop. This is the clearest instance where the AHD loop produces a candidate that is both feasible and measurably better under identical evaluation instances. However, when the comparison is repeated over multiple independently drawn TBs, the advantage disappears and performance becomes nearly indistinguishable from these baselines, indicating that the observed gain is tightly coupled to the in-loop evaluation instances and does not translate into a consistent improvement under refreshed TB draws.

### D. Implications for AHD in stochastic simulators

The contrast between controlled in-loop gains and out-of-loop robustness highlights a key practical pitfall of AHD when applied to stochastic simulators. Fixing the evaluation TBs set reduces score variance and improves selection stability, which can make search progress easier to observe and can accelerate convergence to strong candidates. At the same time, it can bias selection toward heuristics that exploit idiosyncrasies of the fixed instances, especially when improvements are small and concentrated in borderline cases.

In this context, the fact that the best candidate arises from the 1-point, no-exploration, LDPC-grounded setting, while comparable outcomes are also observed under 3- and 6-point evaluation with the same recipe, suggests the following pragmatic guidelines for AHD design tailored to LDPC. First, single-point evaluation reduces per-candidate cost and increases throughput. Second, no-exploration sampling concentrates effort around the strongest observed clusters. Third, LDPC-grounded prompting constrains candidate generation toward the correct functional role.

The combination of the three recommendations above can be effective for quickly reaching a high-quality, baseline-like region of the search space. Achieving robust gains beyond the baselines likely requires additional mechanisms that explicitly promote generalization, such as refreshing evaluation TBs periodically, including a validation split in the scoring protocol, or restricting candidate edits to stability-preserving templates.

## V. Conclusion

This paper analyzed score-trace behavior of LLM-guided AHD for improving the CNU in 5G NR LDPC decoding. We compared a Pure FunSearch-style implementation with an EoH-style implementation and ablated evaluation breadth (1/3/6 configuration points), exploration versus no-exploration parent selection in Pure FunSearch, and generic versus LDPC-grounded prompt framing. Score traces across the tested settings show rapid early progress followed by saturation, but Fig. 2–Fig. 4 demonstrate that baseline-like CNU behavior can be reached across 1/3/6-point evaluation when using no-exploration sampling together with an LDPC-grounded prompt. This indicates that evaluation breadth does not prevent convergence to near-baseline solutions, while prompt grounding and parent-selection policy largely determine whether the search converges or stagnates under a fixed generation budget.

The best-performing discovered CNU emerged under single-point evaluation with no-exploration sampling and an LDPC-grounded prompt, producing a function that is structurally close to `boxplus` and `boxplus-phi` and achieves near-baseline behavior. In a controlled evaluation on the same fixed TB set used during the evolutionary loop, this candidate decoded two additional TBs at the reference configuration point relative to the baselines. Under repeated evaluation with independently drawn TBs, the advantage disappeared and performance became nearly indistinguishable from the baselines, indicating that the observed gain is coupled to the in-loop evaluation instances.

Overall, the results show that AHD can synthesize competitive, baseline-like algorithmic components for LDPC decoding, but that robust improvements require evaluation and validation protocols that explicitly target generalization beyond fixed in-loop instances. Future work should therefore prioritize validation-aware scoring (*e.g.*, refreshed TBs or held-out instances) and constrained search spaces that bias candidate generation toward stability-preserving transformations and repair operators of the CNU.

## Acknowledgment

L.E. Chatzieleftheriou is funded by the European Union, Grant Agreement No. 101155506 (RIXISAC).

## References

[1] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, and et al., "Mathematical discoveries from program search with large language models," *Nature*, vol. 625, no. 7995, pp. 468–475, Jan. 2024.

[2] F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang, "Evolution of heuristics: towards efficient automatic algorithm design using large language model," ser. ICML'24. JMLR.org, 2024.

[3] H. Ye, J. Wang, Z. Cao, and et al., "Reevo: large language models as hyper-heuristics with reflective evolution," ser. NIPS '24, 2024.

[4] S. Yao, F. Liu, X. Lin, Z. Lu, Z. Wang, and Q. Zhang, "Multi-objective evolution of heuristic using large language model," ser. AAAI'25/IAAI'25/EAAI'25, 2025.

[5] Z. Huang, W. Wu, K. Wu, J. Wang, and W.-B. Lee, "Calm: Co-evolution of algorithms and language model for automatic heuristic design," *arXiv preprint arXiv:2505.12285*, 2025.

[6] P. V. T. Dat, L. Doan, and H. T. T. Binh, "Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms," ser. AAAI '25, vol. 39, no. 25, pp. 26 931–26 938.

[7] W. E. Ryan *et al.*, "An introduction to ldpc codes," *CRC Handbook for Coding and Signal Processing for Recording Systems*, vol. 5, no. 2, pp. 1–23, 2004.

[8] J. Hoydis, S. Cammerer, F. A. Aoudia, A. Vem, N. Binder, G. Marcus, and A. Keller, "Sionna: An open-source library for next-generation physical layer research," 2023. [Online]. Available: https://arxiv.org/abs/2203.11854