Autonomous Discovery of Wireless Communications Algorithms

Fayçal Aït Aoudia[1], Member, IEEE, Jakob Hoydis[1], Fellow, IEEE, Sebastian Cammerer[2], Senior Member, IEEE, Gian Marti[3], Member, IEEE, Merlin Nimier-David[3], Nicolas Roussel[3], and Alexander Keller[2]

[1]NVIDIA, 92400 Courbevoie, France
[2]NVIDIA, 10623 Berlin, Germany
[3]NVIDIA, 8004 Zürich, Switzerland

Corresponding author: Fayçal Aït Aoudia (e-mail: faitaoudia@nvidia.com).

Parts of this article were presented at European Wireless, June 2026 [1]. Code repository: https://github.com/nvlabs/the-ai-telco-engineer

arXiv:2607.17762v1 [cs.IT] 20 Jul 2026

ABSTRACT Large language model (LLM)-driven evolutionary search is an emerging algorithm-discovery paradigm that has already produced novel results in several scientific fields. Yet its application to wireless communications remains largely unexplored. To bridge this gap, we introduce The AI Telco Engineer (AITE), a framework to autonomously design algorithms for complex communication problems, while navigating performance–complexity tradeoffs. We showcase AITE on two challenging physical-layer problems: designing an equalizer for an orthogonal time–frequency space (OTFS) system, and constructing a receiver algorithm for an orthogonal frequency-division multiplexing (OFDM) system using a custom constellation and operating without pilots. For the first task, AITE develops algorithms that outperform the best-known solutions while reducing computational latency by a factor of 3.6 compared to the strongest baseline. For the second task, it discovers the first explicit, explainable algorithms that achieve performance parity with state-of-the-art neural receivers. These results demonstrate the strong potential of LLM-driven evolutionary search for the autonomous discovery of next-generation wireless communications algorithms.

INDEX TERMS algorithm discovery, evolutionary search, large language model (LLM), pilotless communication, orthogonal frequency-time space (OTFS), wireless communications

I. Introduction

ALGORITHMS are central to modern communication systems. They largely determine reliability and efficiency, and serve as important product differentiators. Designing and implementing such algorithms is a demanding task for communications system engineers, requiring domain expertise, mathematical insight, and extensive experimentation to identify robust implementations and suitable hyperparameters. Moreover, solutions that perform well analytically or in simulation may behave unexpectedly once integrated into a full system, where finding the root cause may be difficult. Large language models (LLMs) and agentic artificial intelligence (AI) have advanced rapidly, creating an opportunity to dramatically reduce the human effort required for algorithm development. In this article, we therefore investigate their ability to autonomously discover, implement, and optimize wireless communication algorithms.

An algorithm exists at two levels: an abstract mathematical description and a concrete implementation that can be executed. Thus, the optimization of an algorithm can be cast as a programming problem. The target of the problem depends on properties of the concrete implementation, such as its accuracy or latency when run on a specific hardware platform. One way of tackling such problems is genetic programming (GP) [2]. GP relies on evolutionary algorithms to improve computer programs by iteratively applying genetic operators, such as random mutation and crossover, to a generation of seed programs. One of the main problems of GP relates to the definition of the search space, i.e., the set of valid code modifications, which makes scaling to complex programs difficult and requires deep domain expertise.

The integration of LLMs into GP has gradually expanded the field into LLM-driven evolutionary search. The authors of [3] were among the first to replace GP's mutation operator with an LLM that actually understands the code and provides logical program modifications, instead of random edits. This idea was further developed in [4] and [5], where LLMs were also used for crossover operations. The first provably

new scientific results were generated by FunSearch [6], which leveraged a fixed programmatic skeleton restricting the LLM's search to an isolated target function, combined with island-based population management. This idea was scaled to multifile codebases in AlphaEvolve [7], which was reproduced shortly thereafter as the open-source framework OpenEvolve [8]. Since then, there has been a rapid proliferation of LLM-based evolutionary methods for algorithm discovery, such as ShinkaEvolve [9], GEPA [10], and the SkyDiscover framework [11] (including AdaEvolve [12] and EvoX [13]). They introduce several critical capabilities, such as embedding-based novelty rejection, multiarmed bandit LLM selection, adaptive exploration and compute budget control, as well as optimization of the search strategy code itself. Recently, the code-generation step was extended from a single LLM invocation to tool-using coding agents that iteratively implement, evaluate, and refine solutions before submitting them [1], [14]. This extension is particularly important for challenging coding problems that require the use of debugging and profiling tools or documentation lookups.

In this article, we report on initial experiments applying LLM-driven evolutionary search to the design of wireless communications algorithms, providing a useful starting point for others interested in this direction. Notably, even these early results match or surpass the state of the art on the considered tasks, suggesting that agentic AI may have crossed a critical capability threshold, with profound implications for the future of research and development in our field.

The only directly related publications we are aware of are [15] and [16], which apply LLM-driven evolutionary search to the optimization of deletion-correcting codes and antenna port selection in a fluid antenna system, respectively. The authors of [17] describe a versatile platform capable of automating research within a large-scale radio testbed, without providing detailed technical results.

The rest of this article is structured as follows. Section II describes the architecture of our evolutionary search framework called The AI Telco Engineer (AITE), which is available as open-source software [18] and implemented in Python using LangChain [19]. We did not rely on existing frameworks such as [8], [9], [11], although these could have been adapted. In Sections III and IV, we describe two specific physical-layer problems that AITE was tasked with, namely developing an equalizer for an orthogonal time–frequency space (OTFS) system [20] as well as a receiver algorithm for an orthogonal frequency-division multiplexing (OFDM) system operating without pilots and with custom constellations [21]. We choose these two tasks because, although both are nontrivial, they are very different in nature. Equalizer design for OTFS is well studied, so the main challenge is to find efficient implementations that simultaneously improve on existing solutions. The pilotless receiver, in contrast, has essentially no prior art, so the biggest challenge is to discover a working algorithm. For the first task, AITE found algorithms that outperform state-of-the-art equalizers while running about $3.6\times$ faster. For the second task, only solutions based on neural networks were known, and AITE found novel explicit algorithms with comparable performance. Section V concludes the article with a discussion of open problems and future directions. The Appendices provide detailed descriptions of selected algorithms generated by AITE and illustrate the depth and implementation complexity of the discovered solutions.

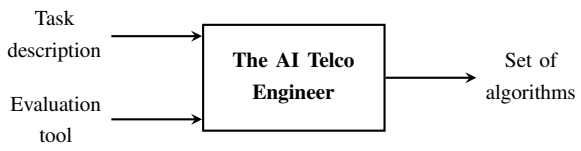


FIGURE 1. AITE from an input–output perspective. The user describes a task, i.e., a problem description paired with an evaluation tool, and AITE returns a set of algorithms that explores the metric–complexity tradeoff.

II. Architecture of The AI Telco Engineer

AITE is an LLM-driven framework that iteratively explores the space of algorithms that solve a specified problem. An input–output view of the system is shown in Fig. 1. AITE takes as input a *task*, defined by a natural-language problem description paired with an immutable, task-specific *evaluation tool*. The evaluation tool executes algorithm implementations and returns a scalar metric evaluating the algorithm's performance and a scalar measuring the complexity of the implementation (e.g., via the measured latency). Because the evaluation tool is not editable by the system and can only be invoked, AITE prevents "metric gaming" in which the scoring procedure is modified to artificially inflate performance.

AITE does not focus on evolving a single best solution, but rather aims to output a *set* of solutions that spans the performance–complexity tradeoff. Concretely, solutions are positioned in the two-dimensional plane spanned by the task metric and the complexity measure returned by the evaluation tool, yielding a point cloud from which the user can pick an operating point after the run, without being involved in the optimization itself. The optimization objective is to expand the Pareto front by iteratively evolving a population of algorithms. The same principle would extend naturally to a higher-dimensional objective space, although we have not explored this in our experiments.

A. Global Architecture

As illustrated in Fig. 2, AITE follows a two-tier architecture inspired by [6], [7], [9]. Most importantly, it relies on agentic code generation as in [1], [14], instead of a single LLM invocation. This architecture proved effective for the tasks considered here. An *orchestrator* drives the global optimization loop, while a pool of *workers* implements and refines solutions in parallel. Optimization proceeds over multiple generations. At each generation, the orchestrator proposes N distinct abstract algorithmic ideas, and a population of M workers is distributed across these ideas, with M/N

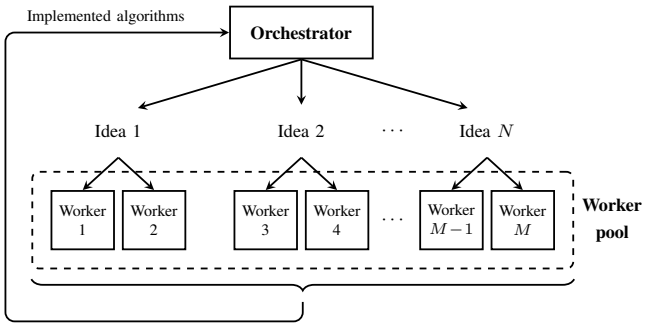


FIGURE 2. Overview of the idea-driven iterative optimization loop. The orchestrator generates ideas that are distributed to parallel workers, each operating in its own workspace. Implemented algorithms flow back to the orchestrator to seed the next generation.

workers assigned to each. Each worker runs independently, in parallel, and within its own containerized environment referred to as a *workspace*. Neither the orchestrator nor the workers have internet access.

Assigning multiple workers to the same idea exploits LLM stochasticity (assuming a nonzero decoding temperature). Different workers implementing the same idea tend to produce different implementations, thereby reducing the risk of discarding a promising idea due to a single poor implementation. After all workers in a generation complete, the orchestrator aggregates the resulting implementations, updates a global *leaderboard*, and uses the accumulated outcomes to propose the next generation of ideas. Subsequent ideas can be entirely novel approaches, refinements of previously explored ones, or combinations thereof, depending on the current state of the performance–complexity frontier.

B. Orchestrator

The orchestrator architecture is shown in Fig. 3. It coordinates the optimization by (i) generating diverse ideas, (ii) dispatching them to workers, (iii) post-processing and summarizing the algorithms produced by the workers, and (iv) maintaining a global leaderboard that records all algorithm implementations and their evaluation scores. The leaderboard groups algorithms by their assigned idea and stores multiple (metric, complexity) evaluation points per algorithm when postrun hyperparameter tuning is enabled (Section II-C1). The Pareto front across all implementations is also maintained.

1) Idea Generation

At each generation, the orchestrator LLM is instructed to produce N *distinct* algorithmic ideas that span the metric–complexity tradeoff. Centralizing this step avoids a scenario where independent workers each choose their own approach and then tend to converge on similar methods, causing the explored space to collapse. Beyond the first generation, the idea budget must be split between *refining* previously explored approaches (exploitation) and proposing *fresh* ones (exploration). Because the number of workers, and hence the per-generation budget, is limited, this allocation is critical to search efficiency [9], [12]. AITE delegates this tradeoff to the orchestrator LLM and instructs it to consistently reserve a part of the budget for fresh ideas throughout the run.

For the initial generation, the orchestrator LLM receives only the user query and is instructed to output exactly N distinct algorithmic ideas, since there are no prior ones to refine. For subsequent generations, idea generation is conditioned on previously explored ideas and their corresponding implemented algorithms. However, providing the entire leaderboard to the orchestrator is not scalable due to the limited context window of the LLM. Instead, AITE uses a two-part context construction: Pareto-front entries are always included, and additional off-front entries are sampled randomly. Off-front sampling includes at most one representative algorithm per idea and excludes algorithms that did not follow the assigned approach, based on a verdict produced by the post-processing stage (cf. Section II-B3).

Off-front entries are sampled without replacement using a temperature-controlled softmax over a spread-normalized metric gap. Let m_i denote the metric of algorithm i in the off-front pool and let $m^\star$ be the best metric in the pool. Define the gap $g_i = |m_i - m^\star|$ and the pool spread $S = \max_i g_i$. The normalized gap is $\tilde{g}_i = g_i/S$. If $S = 0$, we set $\tilde{g}_i = 0$ for all i, resulting in uniform sampling. The sampling probability is then $p_i \propto \exp(-\tilde{g}_i/T)$, where $T > 0$ is a temperature hyperparameter that remains fixed throughout the run. Smaller values of T yield greedier selection, whereas larger values yield nearly uniform sampling.

2) Job Launching

Once a generation's ideas have been generated, the job-launching stage turns them into work units and dispatches them to the worker pool. The N ideas are distributed across the M available workers, with M/N workers assigned to each idea (see Fig. 2). Every worker therefore receives a single idea and is tasked with implementing it in its own dedicated workspace.

For each worker, the orchestrator assembles a *job*: a self-contained initial context that fully specifies what the worker must do. Beyond the assigned idea, this context includes the task description shared by all workers and, when refining ideas, references to one or more earlier implementations that the worker should build upon. Concretely, a job is materialized as a single prompt template into which the task description, the assigned idea, and the source code of optional reference algorithms are injected. The references' achieved metrics are deliberately withheld. Exposing them tends to bias the worker toward small edits of an existing solution rather than implementing the assigned approach.

As detailed in Section II-B4, the prompt template is not fixed but refined across generations to improve worker effi-

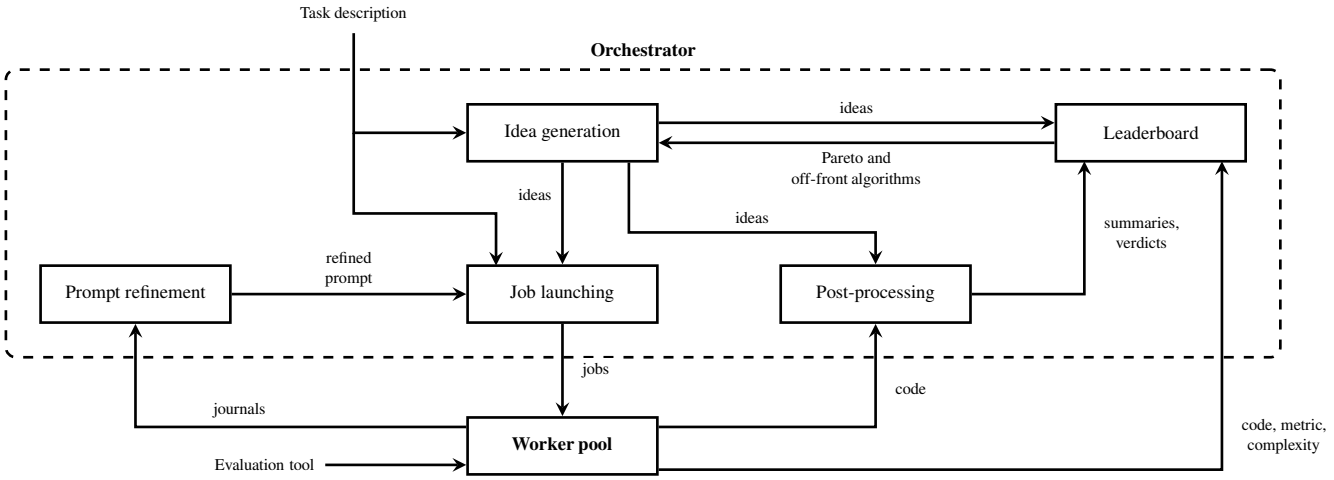


FIGURE 3. **Orchestrator internal structure. Idea generation is conditioned on the task description together with Pareto-front and off-front algorithms drawn from the leaderboard. The job-launching stage dispatches ideas to the worker pool. Workers return code, metric, and complexity to the leaderboard. The post-processing module pairs each implementation with its assigned idea to produce summaries and adherence verdicts, which are also stored in the leaderboard. Worker journals feed the prompt-refinement stage, whose refined prompts are injected into subsequent jobs.**

ciency, while protected placeholders guarantee that the task description and the assigned idea are preserved verbatim.

3) Post-Processing of Generated Algorithms

Worker outputs do not always follow their assigned ideas, especially when an idea is difficult to implement or when the worker LLM fails to adhere to the provided approach description. We empirically observed that state-of-the-art LLMs (e.g., GPT-5.5) diverge from the assigned idea less frequently than smaller open-weight models (e.g., MiniMax-M2.5 or gpt-oss-120b). Regardless of its source, such divergence can mislead the orchestrator, e.g., into discarding an idea that appears to perform poorly when, in fact, it was never actually implemented. To mitigate this, AITE post-processes each completed workspace by reading the resulting code and prompting the orchestrator LLM to produce a concise summary prefixed with a ternary verdict—*yes*, *partial*, or *no*—indicating whether the implemented algorithm follows the assigned idea, independent of its achieved score.

These summaries serve two purposes. First, they provide a compact, human-readable description of what each algorithm actually implements, allowing the orchestrator to reason over a large history of explored methods. Second, they act as a filter when sampling off-front entries for idea generation (see Section II-B1). Only the best-scoring implementations with a *yes* verdict are eligible to represent an idea, preventing the orchestrator from being misled by spurious deviations.

4) Worker Prompt Refinement

AITE is designed to operate with different LLMs, which can vary widely in their tool-use patterns and ability to adhere to instructions. To improve worker efficiency across generations, AITE refines the worker prompt template using a dedicated prompt-refinement stage. After each worker completes, its execution journal (which records all LLM and tool calls together with their outcomes) is read and structured process statistics (e.g., number of evaluation attempts, file reads and writes, and timeouts) are extracted. An LLM is then used to produce a behavioral critique.

At the end of a generation, the orchestrator aggregates these behavioral critiques and requests an LLM to produce a refined prompt template for the next generation, which is used when launching the next generation's jobs (see Section II-B2). For example, if multiple workers make the same coding error, the refined prompt may include a warning to avoid this error in the future. The refinement affects only the generic instructions that govern how a worker operates. The task description and the assigned approach are injected through placeholders and are therefore never altered. This design lets AITE adapt to different LLM behaviors while keeping algorithm selection and exploration decisions centralized in the orchestrator's idea-generation stage.

C. Worker Workflow

Having described the orchestrator, we now turn to the workers that implement the ideas it dispatches. Workers operate in parallel, each running as a separate process that hosts a single LLM-driven ReAct agent [22] which repeatedly reasons about what to do, takes an action using a tool, observes the result, and adapts until it completes the task. The resulting worker architecture is shown in Fig. 4. Tasks are distributed by a worker pool, which submits them to the worker processes and collects their completion results asynchronously. Each task is associated with a fresh workspace, isolated through a container interface to prevent interference across workers and to protect the host environment.

Within a workspace, the worker follows a two-file workflow: `draft.py` is used for iterative development and

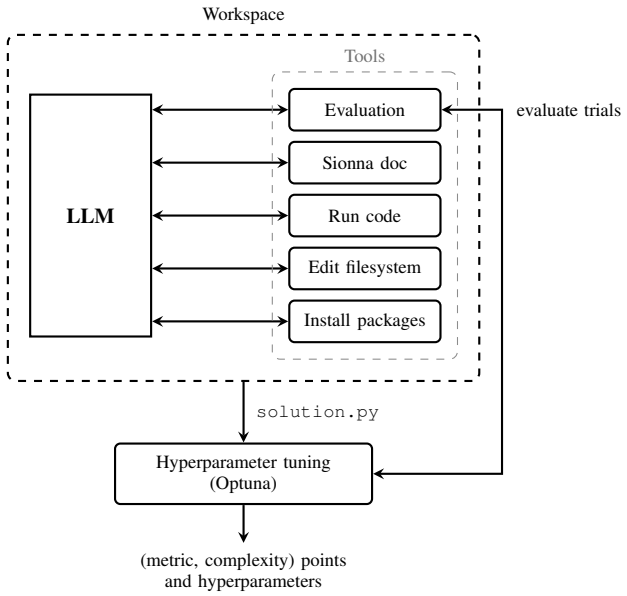


FIGURE 4. Worker architecture. Each worker runs an agentic ReAct loop [22] operating inside an isolated containerized workspace with access to tools, including the evaluation tool. Once the worker completes, an optional postrun stage tunes the algorithm's hyperparameters.

experimentation, while `solution.py` stores the best-performing code encountered so far.[1] The evaluation tool is invoked repeatedly by the worker LLM on `draft.py`. When a new best metric is achieved, the framework automatically copies `draft.py` to `solution.py`, ensuring that progress is preserved even if the worker is interrupted. The worker is terminated either when the worker LLM decides to stop or when a generation-level timeout is reached. In either case, `solution.py` is retained as the worker's output for downstream processing.

1) Hyperparameter Fine-Tuning

To reduce the risk of discarding a promising algorithm due to poorly chosen hyperparameters, AITE performs optional postrun hyperparameter tuning on each successful worker output, as shown in Fig. 4. The worker LLM is instructed to declare tunable hyperparameters by calling one of the following forms:

```
HP.get("name", default, low=..., high=...)
HP.get("name", default, choices=[...])
```

corresponding, respectively, to a bounded numerical range and a categorical choice set. The framework parses the abstract syntax tree of `solution.py` to extract the implied search space, then runs Bayesian multiobjective optimization using Optuna [23] over the task metric and complexity.

Each Optuna trial evaluates `solution.py` for one candidate hyperparameter configuration using the evaluation tool and yields a single (metric, complexity) point. Because the two objectives often conflict, tuning does not return a single best configuration but a set of Pareto-optimal ones. Each retained configuration is then attached to the worker's generated algorithm in the global leaderboard as a distinct (metric, complexity) point, tagged with its hyperparameter values. A single worker output may therefore contribute several leaderboard points, rather than just one.

[1] Although we use Python in this work, having AITE write code in a different language is straightforward.

2) Sionna

In our experiments, AITE uses Sionna [24] to perform link-level simulations. To help workers write correct Sionna code (in PyTorch [25]), the framework provides a documentation tool offering three operations: *Search*, *Help*, and *List*. *Search* performs semantic retrieval over the Sionna application programming interface (API) docstrings and curated tutorials using a FAISS [26] vector store with an external reranker. *Help* retrieves the full docstring and signature of a Sionna symbol, while *List* enumerates available classes and functions within a given Sionna module.

3) Task-Specific Evaluation Tool

The evaluation tool is a critical component of the system. It is supplied by the user as part of the task specification and, by defining the metric and the complexity proxy to optimize, fixes the two objectives that drive the entire search. In our experience, it is also typically the bottleneck of the optimization, consuming the largest share of the compute and time budget. Communication algorithms are usually assessed through Monte Carlo simulations, which trade accuracy for compute. More accurate or exhaustive simulations therefore come at the cost of slower evaluations and, for a fixed time and compute budget, fewer generations of the framework.

In AITE, each task provides its evaluation tool as a pluggable component that executes worker code and returns a standardized, parseable output. The first line follows one of two forms: `SUCCESS, <metric>, <complexity>` or a bare `FAILURE`. A binary success/failure flag is thus always available and, on success, the returned metric and complexity define the algorithm's position in the two-objective plane. Beyond scoring, the evaluation tool can enforce constraints by returning a failure whenever a requirement is violated (e.g., when a secondary metric exceeds a threshold), thereby restricting metric optimization to a feasible set. It can also return error messages and warnings. To prevent any single evaluation from stalling the optimization loop, each invocation is subject to a configurable timeout. Moreover, workers can invoke the evaluation tool but cannot modify it, which prevents direct modification of the scoring procedure while still allowing arbitrary user-defined evaluation code.

III. OTFS Equalizer Design

The first task assigned to AITE is to implement an equalizer for an OTFS system [20]. In short, the receiver makes the

noisy observation $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{w}$ of a vector of constellation symbols $\mathbf{x}$ through a channel matrix $\mathbf{H}$. Knowing $\mathbf{H}$ and the variance N_0 of the elements of the noise vector $\mathbf{w}$, the equalizer must produce bitwise log-likelihood ratios (LLRs) for a channel decoder. Equalizers are ranked by the normalized validation error (NVE) [27], the ratio of the coded block error rate (BLER) achieved by the equalizer to that of a reference equalizer, averaged over a set of signal-to-noise ratio (SNR) values $\mathcal{S}$,

$$\mathrm{NVE} = \frac{1}{|\mathcal{S}|} \sum_{\mathrm{SNR}\in\mathcal{S}} \frac{\mathrm{BLER}_{\mathrm{cand}}(\mathrm{SNR})}{\mathrm{BLER}_{\mathrm{ref}}(\mathrm{SNR})}, \tag{1}$$

so that $\mathrm{NVE} \leq 1$ means the equalizer matches or beats the reference on this averaged metric. The latency of the equalizer (which must be compatible with `torch.compile`) is used as a proxy for complexity.

A. System Model

We consider an OTFS system whose transmit signal is, for every sub-block, an OFDM waveform with a cyclic prefix (CP). The data symbols $x[l,k]$ are defined on the delay–Doppler (DD) grid, for $0 \leq l < M$ and $0 \leq k < N$, where M and N denote the delay and Doppler bin count, respectively. The data symbols are mapped to the time–frequency grid by the inverse symplectic finite Fourier transform

$$X_{\mathrm{tf}}[n,m] = \frac{1}{\sqrt{MN}} \sum_{l=0}^{M-1} \sum_{k=0}^{N-1} x[l,k]\, e^{j2\pi\left(\frac{nk}{N} - \frac{ml}{M}\right)} \tag{2}$$

and then modulated onto a CP-OFDM waveform with a rectangular transmit pulse, CP length T_{cp}, subcarrier spacing Δf, symbol period $T = 1/\Delta f$, and sample period $T_{\mathrm{s}}{=}T/M$. We assume a linear time-varying multipath channel with P paths

$$h(\tau,\nu) = \sum_{i=1}^{P} h_i\, \delta(\tau - \tau_i)\, \delta(\nu - \nu_i), \tag{3}$$

where path i has gain h_i, delay τ_i, and Doppler ν_i. The CP is assumed longer than the delay spread and the path parameters h_i, τ_i, and ν_i are constant over the OTFS frame.

As shown in Appendix A, one can derive the following input–output relation for the OTFS system:

$$\mathbf{y} = \mathbf{H}\,\mathbf{x} + \mathbf{w}, \tag{4}$$

where $\mathbf{x}, \mathbf{y} \in \mathbb{C}^{MN}$ are the vectorized transmitted and received symbols, respectively, $\mathbf{H} \in \mathbb{C}^{MN\times MN}$ is the effective DD channel matrix with elements given by (14), and $\mathbf{w} \in \mathbb{C}^{MN}$ is a vector of complex additive white Gaussian noise (AWGN) with variance N_0. As discussed at greater length in the appendix, one may truncate the channel matrix to a row-sparse approximation $\widetilde{\mathbf{H}}$, e.g., by keeping only the top-k strongest taps per row, with little loss in accuracy. This structure can be efficiently exploited by several classes of equalizers.

Since Sionna [24] does not support OTFS in its current version, we provide a new Sionna OTFS module with all required functionality in the article's code repository.

B. Prior Art

Most practical OTFS equalizers are iterative schemes that operate on the row-sparse $\widetilde{\mathbf{H}}$ channel representation instead of the dense $\mathbf{H}$ matrix. This avoids the prohibitively high complexity of inverting a large $MN \times MN$ matrix, as in, for example, the linear minimum mean-square error (LMMSE) estimate $\widehat{\mathbf{x}} = (\mathbf{H}^{\mathsf{H}}\mathbf{H} + N_0\mathbf{I})^{-1}\mathbf{H}^{\mathsf{H}}\mathbf{y}$. The interference-cancellation message passing (MP) detector of [28] is the canonical example, exchanging beliefs over the discrete constellation symbols on the factor graph induced by the sparse channel. Gaussian-message variants reduce the per-edge cost while retaining calibrated soft outputs, e.g., the unitary approximate message passing (UAMP) detector [29] and the expectation propagation (EP) detector [30]—which serves as the reference baseline in this work due to its excellent performance and low complexity. All of these exploit the row sparsity of $\widetilde{\mathbf{H}}$, so their per-iteration complexity scales as MNS, where S is the number of retained taps per row, rather than as $(MN)^2$. Low-complexity LMMSE receivers in [31], [32] exploit a *full* block-circulant-with-circulant-blocks effective DD channel matrix, which reduces equalization to a 2-D discrete Fourier transform (DFT). That stronger property holds for ideal biorthogonal pulses with integer or slowly varying Doppler, but fails under the rectangular-CP waveform with *continuous* delay *and* Doppler considered here. The channel nevertheless remains block-circulant along the Doppler index [28], so that a Doppler DFT still decouples the $MN \times MN$ system into N independent $M \times M$ blocks. Most competitive algorithms discovered in this task exploit this structure.

C. Simulation Setup

We consider an OTFS system with $M = N = 64$ delay and Doppler bins, a carrier frequency of 4 GHz, a subcarrier spacing of $\Delta f = 15$ kHz, and a 16-sample CP. Information bits are encoded by a rate-$1/2$ 5G low-density parity-check (LDPC) code with block length 16 384, mapped to a 16-quadrature amplitude modulation (QAM) constellation, and then placed on the DD grid. As perfect channel state information (CSI) is assumed, no pilots need to be sent.

We use the continuous DD channel model (13) (see Appendix A) with $P = 6$ paths. Per frame, the path gains are independently and identically distributed according to $\mathcal{CN}(0, 1/P)$, the delays are uniform on $[0, 14\,T_{\mathrm{s}}]$ (i.e., up to 14.6 µs), and the Doppler shifts are uniform on $[-0.5\Delta f, 0.5\Delta f]$ (i.e., from −7.5 kHz to 7.5 kHz). The first path is fixed at zero delay and zero Doppler. The path parameters are constant over the frame and the exact full-period input–output relation is used for the simulation of $\mathbf{y}$. An equalizer can use either the dense representation or a row-sparse version obtained by retaining the top-k strongest taps per row, where the number of retained taps is a free parameter. The noise variance follows the unit-average-symbol-energy convention $N_0 = 10^{-\mathrm{SNR}_{\mathrm{dB}}/10}$, and the evaluation

TABLE 1. Algorithm families explored in the candidate search for the OTFS equalizer design task

Family	Core operation	Candidates	Best NVE	Latency of best
VAMP / OAMP [33], [34]	Block-FFT linear module with extrinsic Gaussian–QAM iterations	557	0.496	5.28 ms
EP / EC [30], [35]	Linear Gaussian updates alternating with extrinsic QAM denoiser	392	0.496	6.40 ms
Proximal MAP / ADMM [36], [37]	Projected-gradient or ADMM steps with QAM proximal thresholding	116	0.549	6.56 ms
List / K-best [38], [39]	Local search over a small QAM neighborhood with residual scoring	88	0.561	3.36 ms
GAMP / AMP [40], [41]	Scalar AMP on the sparse DD graph with Onsager correction	108	0.630	37.60 ms
Linear / LMMSE [31], [32]	One-shot Wiener solve, dense or Doppler-FFT block-structured	252	0.659	2.24 ms
BP / MP [28]	Sparse-graph belief propagation with leave-one-out interference	97	0.677	19.52 ms
PIC / SIC [28]	Parallel or successive interference cancellation on sparse residuals	115	0.677	10.64 ms
Krylov / PCG [42]	Iterative solve of the regularized normal equations	130	1.142	10.64 ms
Fusion / gating [43]	Weighted combination of multiple detector LLR streams	22	1.258	3.28 ms
Low-rank / subspace [44]	Reduced-dimension solve via low-rank channel approximation	13	1.823	5.60 ms

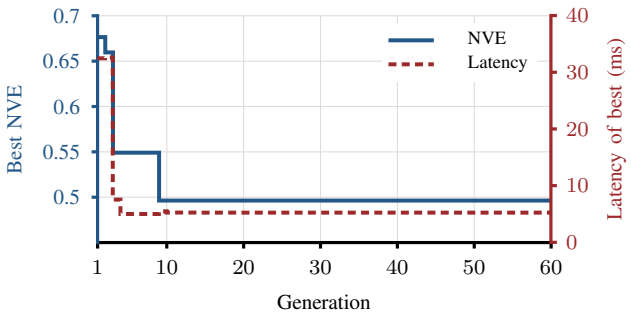
Abbreviations: vector approximate message passing (VAMP), orthogonal approximate message passing (OAMP), expectation propagation (EP), expectation consistency (EC), maximum a posteriori (MAP), alternating direction method of multipliers (ADMM), generalized approximate message passing (GAMP), approximate message passing (AMP), linear minimum mean-square error (LMMSE), belief propagation (BP), message passing (MP), parallel interference cancellation (PIC), successive interference cancellation (SIC), preconditioned conjugate gradient (PCG).

SNR grid for the NVE calculation is $\mathcal{S} = \{13\,\text{dB}, 16\,\text{dB}\}$, with EP [30] (top-$k = 256$) as the reference equalizer.

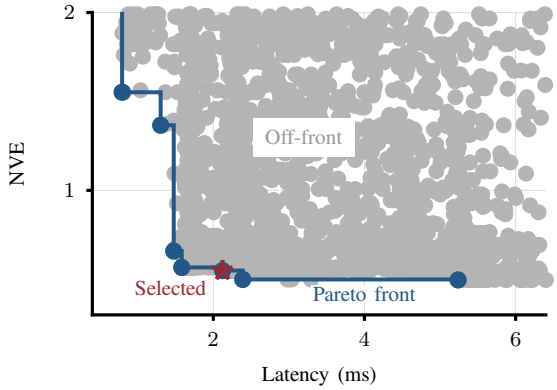
D. Results

Over 60 generations, AITE produced 1890 working candidates implementing 1114 distinct equalizer architectures. We used GPT-5.5 as the underlying LLM and a cluster of 16 graphics processing units (GPUs) running 32 workers per generation. The search broadly explored the accuracy–latency tradeoff across a large variety of algorithm families cataloged in Table 1. Fig. 5(a) shows the evolution of the best NVE across generations and the corresponding latency. The best NVE decreases from 0.677 to 0.549 by generation 3 and reaches 0.496 by generation 9, with no further improvement observed thereafter. The search then focused on ways to reduce latency without degrading performance. Fig. 5(b) shows the Pareto front together with off-front solutions, zoomed to the low-NVE-low-latency region for readability. Notably, many families of algorithms reach comparable performance, especially with the right hyperparameter tuning. All well-performing algorithms exploit the block-circulant channel structure mentioned earlier.

We select a candidate from the expectation consistency (EC) family found in the 59th generation with an interesting accuracy–latency tradeoff (NVE = 0.549, latency 2.17 ms), highlighted in Fig. 5(b), as a representative example for comparison with several baselines. As shown in Fig. 6, the generated algorithm beats all conventional baselines in coded BLER, including MP [28], EP [30], and UAMP [29]. At the same time, Table 2 shows a per-frame latency of 2.17 ms on an NVIDIA RTX PRO 6000 and 3.52 ms on an NVIDIA DGX Spark. These values correspond to speedups of around 3.6× over the most competitive baseline,



(a) Best NVE across generations and their corresponding latency



(b) Generated equalizers in the latency–NVE plane

FIGURE 5. Results of the AITE run for the OTFS equalizer task.

UAMP [29], on both platforms and of 66× and 210× over LMMSE on the RTX PRO 6000 and DGX Spark, respectively. Note that all baselines underwent the same Optuna-based hyperparameter tuning [23] as the generated algorithms, and their implementations were optimized for latency

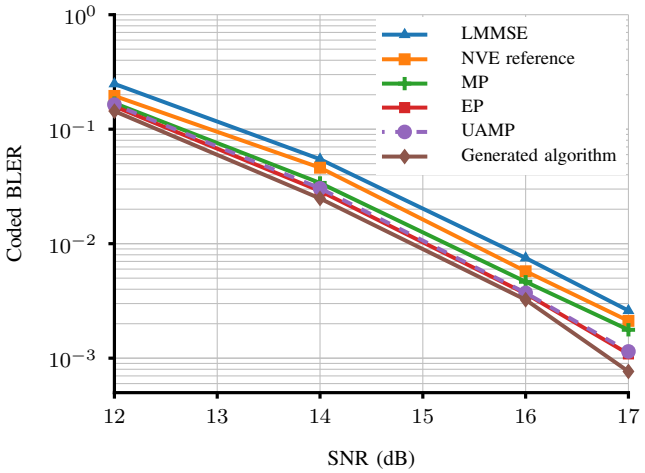


FIGURE 6. **Coded BLER versus SNR for various OTFS equalizers. The values in parentheses show the speedup relative to LMMSE on an NVIDIA RTX PRO 6000. LMMSE uses the dense channel representation; MP uses top-$k = 512$; UAMP uses top-$k = 2048$; EP uses top-$k = 512$; and the generated algorithm uses top-$k = 2560$. The equalizer used as a reference for the NVE in (1) is EP with top-$k = 256$.**

TABLE 2. **Per-frame inference latency. Speedup is relative to LMMSE**

	RTX PRO 6000		DGX Spark	
Equalizer	**Latency**	**Speedup**	**Latency**	**Speedup**
LMMSE	143.84 ms	1.0×	741.92 ms	1.0×
MP	58.56 ms	2.5×	365.92 ms	2.0×
EP	18.64 ms	7.7×	108.00 ms	6.9×
UAMP	7.76 ms	18.5×	12.16 ms	61.0×
Generated	2.17 ms	66.3×	3.52 ms	210.8×

with the help of agentic coding tools. A detailed discussion of the generated algorithm and how it achieves state-of-the-art performance at lower latency is deferred to Appendix B. This fine-grained optimization was performed across hundreds of candidates explored in parallel, illustrating how evolutionary agentic search can apply implementation-level refinement at scale.

IV. Explainable Pilotless OFDM Receiver

The second task assigned to AITE is to generate an explainable pilotless OFDM receiver. This line of work builds on prior results on pilotless OFDM communication [21], which showed that the OFDM waveform can operate without reference signals when the transmitter and receiver are jointly optimized through end-to-end learning. In that setting, a learned constellation shown in Fig. 7 is used to modulate the transmitted bits on all resource elements (REs) of the OFDM resource grid, and a neural-network-based receiver maps the received grid directly to LLRs for the coded bits.

The objective of this experiment is to retain the pilotless transmission scheme using the learned constellation, while replacing the black-box neural receiver with an explainable receiver algorithm discovered by AITE. Algorithms using

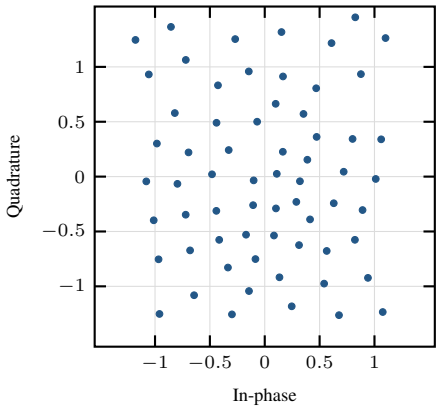
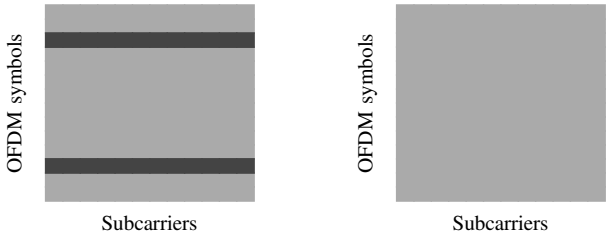


FIGURE 7. **Learned 2^M-point constellation ($M = 6$).**



(a) With DMRS (b) Pilotless (all data)

FIGURE 8. **Resource grids (12×14 REs in this figure) for pilot-assisted and pilotless transmission. Dark REs carry DMRS; light REs carry data. In (a), DMRS symbols are placed in OFDM symbols $t \in \{3, 12\}$.**

sparse pilots exist [45], [46], [47], as do fully pilotless schemes relying on multiple constellations to enable channel estimation [48], [49]. However, the literature still lacks a non-neural pilotless OFDM method that uses a single constellation over all REs. Moreover, none of the existing solutions seems to address the computation of LLRs for the transmitted bits, which are required for channel decoding and are challenging to obtain as they rely on accurate calculation of the channel estimation error. It is thus unlikely that AITE can reproduce an algorithm from its LLM training data.

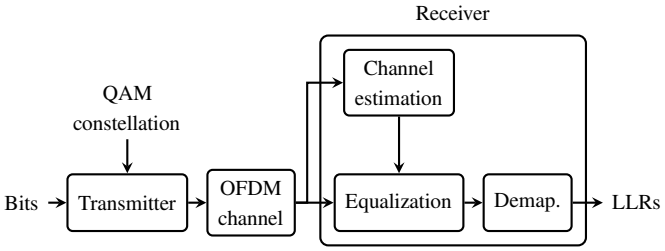
A. System Model

We consider a single-input single-output OFDM link operating on a resource grid with N_F subcarriers and N_T OFDM symbols per slot, yielding $N_F N_T$ REs indexed by (f, t) with $1 \leq f \leq N_F$ and $1 \leq t \leq N_T$. The constellation used to modulate the transmitted bits is denoted by $\mathcal{C} \subset \mathbb{C}$, with $|\mathcal{C}| = 2^M$, where M is the number of bits per symbol.

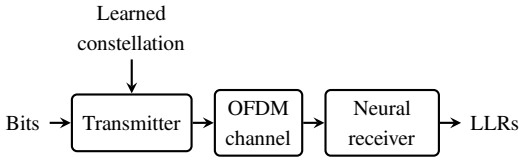
Fig. 8 contrasts the resource allocation in a conventional pilot-assisted grid and in the pilotless grid. In the former case, a subset $\mathcal{P}$ of REs carries known demodulation reference signal (DMRS) symbols. In the latter case, $\mathcal{P} = \emptyset$ and all REs are occupied by data symbols.

On the receiver side, after cyclic-prefix removal and DFT, the complex baseband received symbols satisfy

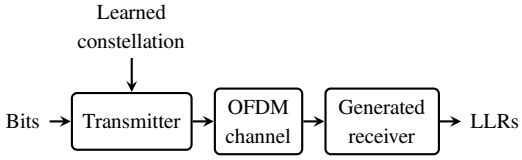
$$\mathbf{Y} = \mathbf{X} \odot \mathbf{H} + \mathbf{W}, \tag{5}$$



(a) Pilot-based baseline



(b) Pilotless with neural receiver



(c) Pilotless with AITE-generated receiver

FIGURE 9. End-to-end OFDM receivers considered in this section: (a) conventional pilot-based baseline with QAM and DMRS; (b) pilotless system with learned constellation and neural receiver [21]; (c) pilotless system with learned constellation and AITE-generated receiver.

where $\mathbf{X} \in \mathbb{C}^{N_F \times N_T}$ are the transmitted symbols, $\mathbf{H} \in \mathbb{C}^{N_F \times N_T}$ is the channel matrix, $\mathbf{W} \in \mathbb{C}^{N_F \times N_T}$ is AWGN with per-RE variance N_0, $\mathbf{Y} \in \mathbb{C}^{N_F \times N_T}$ is the received grid, and $\odot$ denotes elementwise multiplication.

The pilotless system we build upon was introduced in [21], in which the constellation at the transmitter is jointly optimized with a neural network at the receiver, as shown in Fig. 9(b). The neural receiver substitutes for channel estimation, equalization, and demapping. It maps the received resource grid to LLRs. In this work, the neural receiver is a residual convolutional neural network as in [21]. On the transmitter side, only the constellation is learned, while the remainder of the transmission pipeline stays conventional. The trainable parameters consist of a set of 2^M complex numbers $\widetilde{\mathcal{C}} = \{\widetilde{c}_1, \ldots, \widetilde{c}_{2^M}\}$, which are centered and normalized to form the constellation $\mathcal{C}$ used to modulate the transmitted data,

$$\mathcal{C} = \left\{ \frac{\widetilde{c} - \mu}{\sqrt{\frac{1}{2^M}\sum_{\widetilde{c}' \in \widetilde{\mathcal{C}}} |\widetilde{c}'|^2 - |\mu|^2}} \;\middle|\; \widetilde{c} \in \widetilde{\mathcal{C}} \right\}, \qquad (6)$$

where $\mu = \frac{1}{2^M}\sum_{\widetilde{c} \in \widetilde{\mathcal{C}}} \widetilde{c}$, so that $\mathcal{C}$ has zero mean and unit average power. Centering prevents the learned constellation from embedding a superimposed pilot in its mean, which would act as a known reference signal. The learned constellation is applied to every RE of the resource grid, as shown in Fig. 8(b). Consequently, the resulting scheme uses neither orthogonal nor superimposed pilots.

The results reported in [21] show that the neural receiver is able to recover the transmitted bits despite the lack of pilots. Removing pilots frees the REs that conventional systems reserve for reference signals and reallocates them to data, which translates into significant throughput gains. Achieving this requires the learned constellation to adopt an unconventional geometry, shown in Fig. 7. While effective, the neural receiver inherits the general lack of explainability of neural networks. This makes the design difficult to analyze and extend, e.g., to multiple spatial streams or different modulation orders. It also motivates the search for an explainable algorithmic receiver that AITE is asked to find.

B. Simulation Setup

We consider an OFDM system with $N_F = 72$ subcarriers and $N_T = 14$ OFDM symbols. To train the end-to-end learning system (Fig. 9(b)) and generate algorithms with AITE (Fig. 9(c)), we use the TDL-C channel model with a nominal delay spread of 100 ns, a carrier frequency of 2.6 GHz, a subcarrier spacing of 30 kHz, and user speeds between 0 and 3 m/s. The constellation optimized through end-to-end learning, shown in Fig. 7, is reused unchanged for generating algorithms with AITE. The evaluation tool returns the NVE defined in (1), computed over two SNR points, 15 dB and 20 dB, using the pilot-based baseline in Fig. 9(a) as the reference. The average latency required to run the algorithm is used as a proxy for complexity.

We benchmark the generated algorithm on the TDL-A and TDL-D models, which are non-line-of-sight (NLoS) and line-of-sight (LoS) models, respectively, with the same nominal delay spread and user speeds. Evaluating on channel models unseen during training probes the generalization capabilities of the receivers. For all receivers considered in this section, the resulting LLR vector is passed to the same LDPC decoder with code rate $r_c = 0.7$, which outputs hard decisions on the transmitted information bits.

C. Results

We ran the framework for 42 generations with GPT-5.5 as the LLM and 32 workers per generation on a cluster of 16 GPUs. Fig. 10(a) reports the evolution of the search. The best NVE decreases rapidly during the first generations and reaches 1.02 at generation 25; among tied-best solutions, latency continues to decrease through generation 34. In total, AITE produced 1240 functional receivers. Rather than the monotone refinement of a single algorithm, the search can be described as a broad exploration that gradually concentrated around a small number of ideas. The first generation instantiated more than a dozen different receiver families, most of them performing poorly.

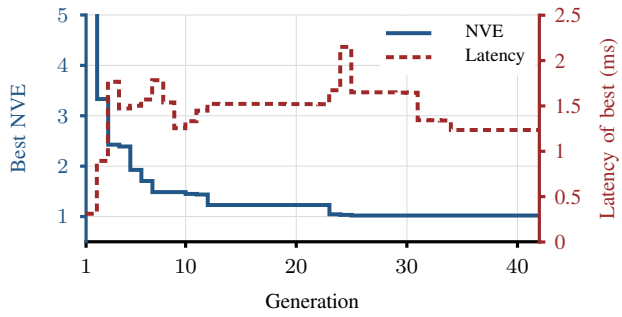
Compared to the OTFS equalizer task, this task proved more challenging for AITE, requiring more than 20 generations to converge, as shown in Fig. 10(a). Since no directly applicable prior art is available, AITE had to discover the key methods enabling blind detection, in what

resembles a research process. Progress originated from a small number of breakthroughs rather than from incremental tuning of a single approach. The first key realization was that the structure of the learned constellation $\mathcal{C}$ can be leveraged to initialize the channel estimate blindly. Because the transmitted symbols are drawn from $\mathcal{C}$ on every RE, the local empirical moments of the received grid relate to those of the constellation, i.e., over a small time–frequency neighborhood, $\mathbb{E}[Y^m] \approx H^m \, \mathbb{E}[C^m]$ for integer m, where C is uniformly distributed on $\mathcal{C}$. Estimating these moments and taking the mth root therefore yields a per-RE channel estimate up to a discrete m-fold phase ambiguity, with the residual ambiguity resolved by selecting the root that maximizes the likelihood $\sum_{c\in\mathcal{C}} \exp\left(-|Y - Hc|^2 / N_0\right)$, which is possible only because the learned constellation, unlike QAM, has no rotational symmetry. The resulting estimate is accurate enough to seed an expectation–maximization (EM) refinement loop that alternates between soft symbol posteriors and a channel update. This single idea moved the receiver out of the failure regime, and essentially every subsequent frontier solution descends from it.

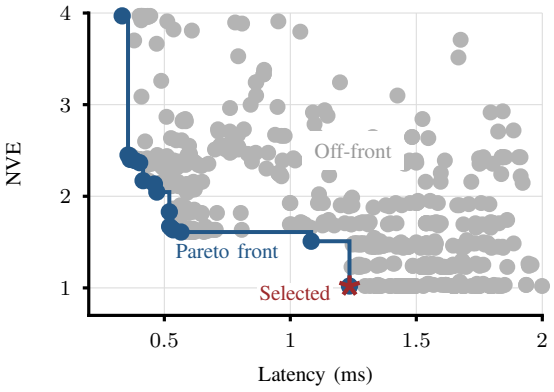
A second breakthrough was not to commit to a single channel estimate, but to retain a small set of competing channel hypotheses per RE and use a log-sum-exp demapper that marginalizes the likelihood over both the candidate channels and the constellation points when computing the per-bit LLRs. From these two ideas, the improvements that mattered were obtained by stacking complementary mechanisms onto this backbone rather than by replacing it. Several of the families that had failed as standalone receivers reappeared at the Pareto front by being injected as refinements.

These auxiliary refinements are of three kinds. First, a belief-propagation refinement exploits the similarity of the channel gains at neighboring REs. These gains largely cancel in the ratio of the two received samples, leaving a relationship governed primarily by the two transmitted constellation points. Treating these pairwise relationships as soft constraints and propagating them across the grid sharpens the symbol decisions where the channel is hard to estimate directly. Second, a decision-directed channel-tracking candidate turns the receiver's soft estimate of the transmitted symbols into virtual pilots, from which an additional channel estimate is formed and smoothed. Third, a local affine model assumes that the channel varies approximately linearly within a small time–frequency patch and fits an affine plane to each patch to denoise the channel estimate.

In summary, after broad, divergent exploration, the lowest-NVE solutions share a common receiver backbone, with the bulk of the gain explained by a few enabling ideas and by the recombination of mechanisms that were individually insufficient. Fig. 10(b) shows the Pareto front of the set of generated receivers together with the off-front solutions, with the view zoomed in on the low-NVE region. Several algorithms attain the lowest NVE of 1.02. Among them, we select the one with the lowest latency, highlighted in Fig. 10(b), for



(a) Best NVE achieved so far and their corresponding latency



(b) Generated algorithms in the latency–NVE plane

FIGURE 10. Results of the AITE run for the pilotless receiver task.

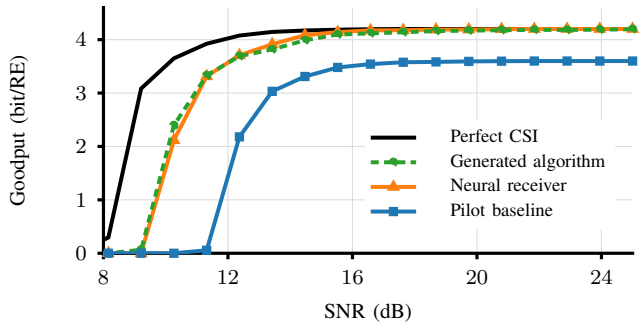
evaluation against the baselines. A more detailed description of this algorithm is provided in Appendix C.

We benchmark the selected algorithm against two baselines: a pilot-based baseline (Fig. 9(a)) and the neural receiver trained jointly with the constellation for pilotless communication (Fig. 9(b)). The pilot-based baseline follows a standard processing chain, i.e., least-squares channel estimation at the pilot positions $\mathcal{P}$, linear interpolation of the channel estimates and error variances onto all data REs, LMMSE equalization, and a posteriori probability demapping to LLRs. This baseline is implemented with the corresponding components from Sionna [24] and employs a 2^M-QAM constellation.

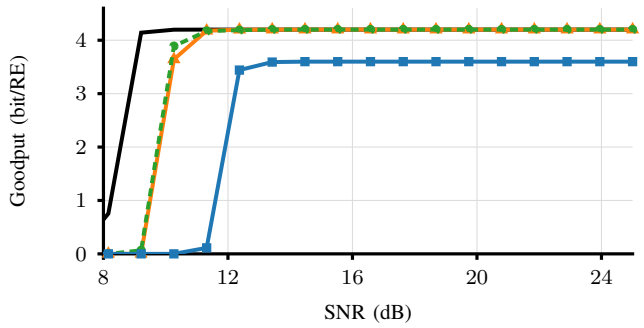
The goodput achieved by the evaluated receivers on TDL-A and TDL-D is reported in Fig. 11(a) and 11(b), respectively, and is defined as

$$\text{Goodput} = (1 - \text{BLER})\, r_c\, r_p\, M, \tag{7}$$

where $r_c = 0.7$ is the code rate and $M = 6$ is the number of bits per constellation symbol. The factor r_p is the fraction of REs carrying data: $r_p = 1$ for the pilotless approaches and the perfect-CSI upper bound, and $r_p = 12/14$ for the pilot-based baseline. As the figures show, the generated algorithm achieves goodput on par with that of the neural receiver on both channel models. Both pilotless receivers realize the expected throughput gain over the pilot-based baseline, which comes from reallocating pilots to data.



(a) Goodput on TDL-A (NLoS)



(b) Goodput on TDL-D (LoS)

FIGURE 11. Evaluation of the selected AITE-generated receiver algorithm against the pilot-based baseline and the neural receiver. The perfect-CSI upper bound uses 2^M-QAM on every RE, exact channel coefficients, and the same demapper and LDPC decoder as the pilot-based baseline.

The latency, measured on an NVIDIA RTX PRO 6000, is 0.694 ms for the neural receiver and 1.23 ms for the generated algorithm. The longer latency is somewhat expected because, with the used PyTorch backend, the convolutional layers of the neural receiver map efficiently onto the GPU, whereas the generated algorithm relies on non-neural signal-processing operations that are less amenable to straightforward parallelization. Being explainable, however, the generated algorithm is easier to analyze and improve. For instance, it could be extended to other modulation orders and multiple spatial streams, or accelerated through a dedicated CUDA implementation. Both are left to future work.

V. Concluding Discussion

Our experiments demonstrate that LLM-driven evolutionary search can discover novel wireless algorithms that are on par with or even surpass the current state of the art. Such methods allow for the parallel exploration and implementation of a large number of algorithmic ideas for a specific problem with the goal of building a performance–complexity Pareto front of solutions from which the most suitable can be selected. We believe that agentic tools like AITE are likely to become essential for the design of competitive algorithms in the future, and the broader impact of this development has been discussed by some of us [50]. Developing AITE led us to identify several practical enablers and limitations, which we discuss in the remainder of this section.

A. Impact of the Underlying LLM

The quality of the generated algorithms is strongly affected by the capabilities of the LLMs used. Experiments with smaller, less capable LLMs yielded significantly less competitive algorithms on complex tasks and required a larger number of iterations to converge. Even when smaller LLMs proposed strong ideas, they often struggled to implement them correctly or to follow the instructions provided by the orchestrator, resulting in a higher rate of unexploitable algorithms. This observation motivated the use of a frontier model (GPT-5.5) for the experiments reported in this article. Each task took approximately four days on a cluster of 16 GPUs and incurred an LLM inference cost of about USD 2500 at the time of writing. However, this cost is likely to decrease significantly in the future. Quantifying the run-to-run variability of the search outcome remains open.

B. The Importance of the Evaluation Tool

The definition of the evaluation metric is one of the most important design decisions because evolutionary search will relentlessly try to optimize for it and, as in supervised machine learning, there is a risk of overfitting to the evaluation tool (including channel models and target hardware platforms). One way to circumvent this could be a *validation tool* whose outputs would only be used to detect when algorithms start to overfit so that they are not selected for refinement in future generations. In addition, we observed that AITE spends most of the running time evaluating candidate algorithms, as opposed to generating and implementing them. For the two tasks considered in this article, this evaluation is based on Monte Carlo simulations of BLER curves for the NVE as well as latency measurements of PyTorch-compiled code. The more accurate the desired metric estimates, the longer the evaluation takes. Designing an efficient evaluation tool is therefore a critical enabler for the scalability of evolutionary search methods. This becomes even more important when evaluations are carried out in large digital twin networks or with hardware-in-the-loop requiring time-consuming compilation or high-level synthesis.

C. Hyperparameter Tuning

Most generated algorithms have hyperparameters, whose number typically grows as the evolutionary search progresses and the algorithms become more complex. As described in Section II-C1, AITE provides a hyperparameter fine-tuning stage, which was applied to every candidate algorithm in the experiments reported here. However, Bayesian optimization becomes impractical with even a few dozen hyperparameters. One solution would be to replace Bayesian optimization with gradient descent for differentiable continuous hyperparameters, combined with a separate optimization method for those defined in discrete spaces, such as the number of iterations. A different approach would be to run substantially fewer generations of the framework and to perform more

exhaustive hyperparameter fine-tuning for each candidate. The optimal allocation problem remains unresolved.

D. Idea Generation: Exploration Versus Exploitation

Evolutionary search frameworks like AITE need to solve an exploration–exploitation tradeoff at the start of each generation, when a finite number of workers must be tasked either with exploring novel ideas or refining ideas from previous generations. Insufficient exploration may cause potential breakthrough ideas to be missed, while insufficient exploitation may cause promising ideas to be discarded prematurely. In AITE, the orchestrator LLM is responsible for solving this tradeoff through careful prompt refinement. Other frameworks adopt different strategies [9], [11], [12]. Improving the sample efficiency of the search is an important direction for future work that would also allow reallocating resources to hyperparameter tuning, cf. Section V-C.

E. Measuring Complexity

We can characterize algorithmic complexity in many ways. Theoretical complexity concepts such as floating-point operations or asymptotic analysis are important for high-level comparisons, but they are often unreliable indicators of actual latency or throughput on target hardware, such as a GPU or a central processing unit. Unfortunately, measuring these concrete metrics requires direct access to the target hardware, which is often infeasible in large compute clusters. Frameworks like AITE can in principle use arbitrary complexity metrics, but, much like the evaluation metric, these metrics must be chosen carefully to properly steer the search toward practically relevant outcomes.

Appendix A
OTFS Input–Output Relation

Based on the system model in Section III-A, we derive a closed-form input–output relation for the OTFS system for use in the evaluation tool's Monte Carlo simulations.

Let us introduce the normalized DD coordinates

$$\widetilde{l}_i \triangleq \tau_i M\Delta f = l_i+\varepsilon_i, \quad \widetilde{k}_i \triangleq \nu_i N(T+T_{\mathrm{cp}}) = k_i+\kappa_i, \tag{8}$$

with nearest integers $l_i, k_i \in \mathbb{Z}$ and fractional parts $\varepsilon_i, \kappa_i \in (-\frac{1}{2}, \frac{1}{2}]$. The received signal $y[l,k]$ on the DD grid after matched filtering, sampling at rate $1/T_{\mathrm{s}}$, CP removal, and the unitary per-sub-block M-point DFT and symplectic finite Fourier transform is then given by (13), where

$$\Phi_i(l) = e^{-j2\pi\nu_i\tau_i}\, e^{j2\pi\nu_i l T_{\mathrm{s}}}, \tag{9}$$

$$\mathcal{D}_Q(\xi) = \frac{1}{Q}\sum_{q=0}^{Q-1} e^{j2\pi q\xi/Q} \tag{10}$$

are per-path phase terms and Q-point Dirichlet kernels, respectively. Here, $(\cdot)_Q$ denotes reduction modulo Q, and $w[l,k]$ is standard complex AWGN with variance N_0. The values of γ_τ and γ_ν run over the windows

$$\mathcal{W}_\tau = \{-\lfloor M/2\rfloor, \ldots, \lceil M/2\rceil - 1\}, \tag{11}$$

$$\mathcal{W}_\nu = \{-\lfloor N/2\rfloor, \ldots, \lceil N/2\rceil - 1\} \tag{12}$$

of size exactly M and N for either parity.

Collecting the DD symbols into vectors $\mathbf{x}, \mathbf{y}, \mathbf{w} \in \mathbb{C}^{MN}$ by assigning each cell (l,k) the scalar index $n(l,k) \triangleq k + Nl$ (row-by-row stacking, with the Doppler index varying fastest, so that $[\mathbf{x}]_{n(l,k)} = x[l,k]$, and likewise for $\mathbf{y}$ and $\mathbf{w}$), we can write the input–output relation (13) compactly as

$$\mathbf{y} = \mathbf{H}\,\mathbf{x} + \mathbf{w},$$

where $\mathbf{H} \in \mathbb{C}^{MN\times MN}$ is the effective DD channel matrix with elements given by (14). For the full windows $\mathcal{W}_\tau, \mathcal{W}_\nu$, the summations over the Kronecker deltas have exactly one nonzero term for every l, l', k, k', so that $\mathbf{H}$ is generally dense for noninteger delays or Dopplers.

In practice, the windows can be truncated with little loss in accuracy to $\widetilde{\mathcal{W}}_\tau = \{-N_\tau, \ldots, N_\tau\}$ and $\widetilde{\mathcal{W}}_\nu = \{-N_\nu, \ldots, N_\nu\}$ with $2N_\tau{+}1 \ll M$ and $2N_\nu{+}1 \ll N$. When these truncated windows are used in (14), only the $(2N_\tau + 1)(2N_\nu + 1)$ columns nearest the integer shift $(l - l_i, k - k_i)$ are nonzero for each path, yielding a row-sparse matrix $\widetilde{\mathbf{H}}$ with at most $S = P\,(2N_\tau + 1)(2N_\nu + 1)$ nonzero entries per row.

Alternatively, rather than fixing the windows a priori, one can compute the exact dense $\mathbf{H}$ and retain in each row only the top-k entries with the largest absolute value. This yields a controlled sparse approximation whose accuracy increases monotonically with k, and—in contrast to the fixed window above—adapts to the actual location of the dominant taps, which is advantageous under pronounced fractional-delay or fractional-Doppler leakage. Equalizers can choose to work with such truncated channel matrices for efficiency, whereas the simulation of $\mathbf{y}$ always uses the full channel matrix.

Appendix B
Discussion of an OTFS Equalizer Candidate

For an observation $\mathbf{y}$ and perfect channel knowledge $\mathbf{H}$, the equalizer's goal is to compute the posterior marginals $p(x_r|\mathbf{y})$ for $r \in \{0, \ldots, MN-1\}$ under the channel likelihood (4) and a discrete prior $p(x_r)$, which accounts for the fact that all symbols are drawn from a QAM constellation $\mathcal{Q} = \{\alpha_1, \ldots, \alpha_Q\}$ of order Q. Computing the joint posterior $p(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$ is prohibitive because all symbols are discrete and coupled by the matrix $\mathbf{H}$. The algorithm sidesteps this by approximating the posterior $p(\mathbf{x}|\mathbf{y})$ with a Gaussian $q(\mathbf{x})$ whose marginal moments are made consistent between two simpler views of the problem, iterating until both agree. This two-module moment-matching construction is an EC approximation [35] and works as follows:

A. Module A: Linear Model with Gaussian Prior

The first view corresponds to the linear model (4) with a Gaussian symbol prior $p_B(\mathbf{x})$ which it receives from Module B. This prior is defined by its natural parameters, the precision $\mathbf{p}_B \in \mathbb{R}_+^{MN}$ and precision-weighted mean $\boldsymbol{\eta}_B \in \mathbb{C}^{MN}$. In the first iteration this message is initialized to the unit-energy Gaussian prior, $\mathbf{p}_B = \mathbf{1}$ and $\boldsymbol{\eta}_B = \mathbf{0}$,

$$y[l,k] = \sum_{i=1}^{P} h_i\, \Phi_i(l) \sum_{\gamma_\tau \in \mathcal{W}_\tau} \sum_{\gamma_\nu \in \mathcal{W}_\nu} \mathcal{D}_M(\gamma_\tau - \varepsilon_i)\mathcal{D}_N(\gamma_\nu + \kappa_i)\, x[(l - l_i - \gamma_\tau)_M\,,\, (k - k_i + \gamma_\nu)_N] + w[l,k] \tag{13}$$

$$[\mathbf{H}]_{n(l,k),\, n(l',k')} = \sum_{i=1}^{P} h_i\, \Phi_i(l) \sum_{\gamma_\tau \in \mathcal{W}_\tau} \mathcal{D}_M(\gamma_\tau - \varepsilon_i)\delta[l' - (l - l_i - \gamma_\tau)_M] \sum_{\gamma_\nu \in \mathcal{W}_\nu} \mathcal{D}_N(\gamma_\nu + \kappa_i)\, \delta[k' - (k - k_i + \gamma_\nu)_N] \tag{14}$$

so that Module A reduces to a plain linear (LMMSE-type) estimate. Under this assumption, the corresponding posterior is also Gaussian with natural parameters $\tilde{\mathbf{p}}_A$ and $\tilde{\boldsymbol{\eta}}_A$:

$$\tilde{\mathbf{p}}_A = \mathbf{1} \oslash \mathrm{diagpart}\left(\mathbf{A}^{-1}\right) \tag{15}$$

$$\tilde{\boldsymbol{\eta}}_A = \tilde{\mathbf{p}}_A \odot \left[\mathbf{A}^{-1}\left(\boldsymbol{\eta}_B + \frac{1}{N_0}\mathbf{H}^{\mathsf{H}}\mathbf{y}\right)\right] \tag{16}$$

$$\mathbf{A} = \mathrm{diag}\left(\mathbf{p}_B\right) + \frac{1}{N_0}\mathbf{H}^{\mathsf{H}}\mathbf{H}, \tag{17}$$

where $\odot$ and $\oslash$ denote elementwise multiplication and division, respectively. Module A sends to Module B only the extrinsic Gaussian with natural parameters

$$\mathbf{p}_A = \tilde{\mathbf{p}}_A - \mathbf{p}_B, \quad \boldsymbol{\eta}_A = \tilde{\boldsymbol{\eta}}_A - \boldsymbol{\eta}_B. \tag{18}$$

B. Module B: QAM Denoiser via Moment Matching

This view treats the discrete prior exactly and accounts for the channel only through the message it receives from Module A, which it interprets as a per-symbol Gaussian pseudo-observation of the transmitted symbols

$$\mathbf{r}_A = \mathbf{x} + \mathbf{n}, \tag{19}$$

where $\mathbf{r}_A = \boldsymbol{\eta}_A \oslash \mathbf{p}_A$, and $\mathbf{n} \sim \mathcal{CN}(0, \mathrm{diag}\left(\mathbf{p}_A\right)^{-1})$. Assuming equiprobable constellation symbols, it then computes for each symbol index r the discrete posterior $\gamma_{r,q} = p(x_r = \alpha_q | r_{A,r})$ over the Q constellation points:

$$\gamma_{r,q} = \frac{\exp\left(-p_{A,r}\,|r_{A,r} - \alpha_q|^2\right)}{\sum_{q'=1}^{Q} \exp\left(-p_{A,r}\,|r_{A,r} - \alpha_{q'}|^2\right)}. \tag{20}$$

As this tilted posterior is not Gaussian, Module B projects it back onto a Gaussian by matching its first two moments, giving the posterior mean $\tilde{\mathbf{r}}_B$ and variance $\tilde{\mathbf{v}}_B$ with

$$\tilde{r}_{B,r} = \sum_{q=1}^{Q} \gamma_{r,q}\, \alpha_q, \quad \tilde{v}_{B,r} = \sum_{q=1}^{Q} \gamma_{r,q}\, |\alpha_q|^2 - |\tilde{r}_{B,r}|^2\,, \tag{21}$$

which lead to the natural parameters $\tilde{\mathbf{p}}_B = \mathbf{1} \oslash \tilde{\mathbf{v}}_B$ and $\tilde{\boldsymbol{\eta}}_B = \tilde{\mathbf{p}}_B \odot \tilde{\mathbf{r}}_B$. Module B then forms the extrinsic message

$$\mathbf{p}_B^{\mathrm{ext}} = \tilde{\mathbf{p}}_B - \mathbf{p}_A, \quad \boldsymbol{\eta}_B^{\mathrm{ext}} = \tilde{\boldsymbol{\eta}}_B - \boldsymbol{\eta}_A \tag{22}$$

and sends a damped version back to Module A:

$$\mathbf{p}_B \leftarrow (1-\beta)\,\mathbf{p}_B + \beta\,\mathbf{p}_B^{\mathrm{ext}}, \quad \boldsymbol{\eta}_B \leftarrow (1-\beta)\,\boldsymbol{\eta}_B + \beta\,\boldsymbol{\eta}_B^{\mathrm{ext}}, \tag{23}$$

with $\beta \in (0, 1]$. This exchange is repeated until the marginals of the two modules agree. In this generic form, the per-bit LLRs are read off from the final symbol posteriors $\gamma_{r,q}$.

C. Optimized Implementation

The generated algorithm is a performance- and latency-optimized version of this algorithm based on the following key modifications:

1) **Per-Doppler block-diagonalization.** The input–output relation (13) is a block-circulant convolution along the Doppler index which can be diagonalized by an N-point DFT along the Doppler axis [28]. This decouples the $MN \times MN$ system into N independent $M \times M$ blocks which can be treated in parallel. The implementation further shares a single precision per delay across all Doppler bins, $\mathbf{p}_B \in \mathbb{R}_+^M$ (the mean $\boldsymbol{\eta}_B \in \mathbb{C}^{M \times N}$ stays per cell), so the same $\mathrm{diag}(\mathbf{p}_B)$ regularizes every block.
2) **No matrix inversion.** Module A never forms $\mathbf{A}^{-1}$. In (16), Module A uses Cholesky factorization followed by forward/back substitution to obtain $\tilde{\boldsymbol{\eta}}_A$ for each Doppler block. The posterior precisions in (15) are approximated by a second-order Jacobi estimate of $\mathrm{diagpart}(\mathbf{A}^{-1})$, averaged over Doppler to one variance per delay index.
3) **Few iterations plus one Richardson half-step.** The algorithm runs for only two full EC iterations. In the third, the Cholesky solve to compute (16) is replaced by a single low-complexity damped Jacobi (i.e., diagonally preconditioned Richardson) update [42]:

$$\mathbf{x}_{\mathrm{R}} = \tilde{\mathbf{r}}_B + \omega\ \mathrm{diag}(\mathbf{A})^{-1}\left(\boldsymbol{\eta}_B + \tfrac{1}{N_0}\mathbf{H}^{\mathsf{H}}\mathbf{y} - \mathbf{A}\,\tilde{\mathbf{r}}_B\right),$$

with relaxation factor ω. The associated precision $\tilde{\mathbf{p}}_A$ is obtained from (15), and the resulting *Richardson cavity* follows by the same extrinsic subtraction as in (18), now with $\mathbf{x}_{\mathrm{R}}$ in place of the exact posterior mean,

$$\mathbf{p}_A^{\mathrm{R}} = \tilde{\mathbf{p}}_A - \mathbf{p}_B, \quad \mathbf{r}_A^{\mathrm{R}} = \left(\tilde{\mathbf{p}}_A \odot \mathbf{x}_{\mathrm{R}} - \boldsymbol{\eta}_B\right) \oslash \mathbf{p}_A^{\mathrm{R}}. \tag{24}$$

This cavity is not fed back to Module B; it is passed directly to the final demapper.
4) **Fused demapping from multiple log-metrics.** Let $\mathcal{L}_{r,q}(\mathbf{r}, \mathbf{p}) = -p_r\,|r_r - \alpha_q|^2$ denote the constellation log-metric of a Gaussian pseudo-observation $(\mathbf{r}, \mathbf{p})$, i.e., the numerator exponent of $\gamma_{r,q}$ in (20). Rather than demapping a single posterior, the implementation linearly fuses three such metrics, evaluated at the Richardson cavity $(\mathbf{r}_A^{\mathrm{R}}, \mathbf{p}_A^{\mathrm{R}})$, the last cavity $(\mathbf{r}_A, \mathbf{p}_A)$,

and the last linear posterior $(\tilde{\boldsymbol{\eta}}_A \oslash \tilde{\mathbf{p}}_A, \tilde{\mathbf{p}}_A)$,

$$\mathcal{L}^{\mathrm{f}}_{r,q} = w_c\, \mathcal{L}_{r,q}(\mathbf{r}^{\mathrm{R}}_A, \mathbf{p}^{\mathrm{R}}_A) + w_t\, \mathcal{L}_{r,q}(\mathbf{r}_A, \mathbf{p}_A) + w_\ell\, \mathcal{L}_{r,q}(\tilde{\boldsymbol{\eta}}_A \oslash \tilde{\mathbf{p}}_A, \tilde{\mathbf{p}}_A) \tag{25}$$

and reads the per-bit LLRs by max-log demapping,

$$\lambda_{r,k} = \max_{q \in \mathcal{S}^1_k} \mathcal{L}^{\mathrm{f}}_{r,q} - \max_{q \in \mathcal{S}^0_k} \mathcal{L}^{\mathrm{f}}_{r,q}, \tag{26}$$

where $\mathcal{S}^b_k = \{q : \text{the } k\text{th label bit of } \alpha_q \text{ is } b\}$.

5) **Scalings and precomputation.** The noise variance is scaled, $\tilde{N}_0 = s_{N_0} N_0$, to correct for model mismatch and used in place of N_0. The data terms $\tilde{N}_0^{-1}\mathbf{H}^{\mathsf{H}}\mathbf{H}$ and $\tilde{N}_0^{-1}\mathbf{H}^{\mathsf{H}}\mathbf{y}$ are formed once per frame and reused across iterations, leaving only $\mathrm{diag}(\mathbf{p}_B)$ and the prior-pull term in (16) to update. The output LLRs are finally rescaled by a factor s_λ before the LDPC decoder.
6) **Hyperparameters.** The tuned hyperparameters comprise the channel sparsity, numerical clipping thresholds, relaxation and damping factors, fusion weights, noise and LLR scalings, and final demapping rule. They are selected by the framework's postrun hyperparameter optimization (Section II-C1).

Appendix C
Discussion of a Pilotless OFDM Receiver Candidate

The receiver maps the received grid $\mathbf{Y}$ to LLRs through a sequence of stages. For clarity, we only describe the key operation of each stage and omit details such as windowing, damping, and reliability gating. The complete implementation is available in the code repository [18]. Throughout, $\langle\cdot\rangle_{f,t}$ denotes a local average over a small time–frequency window centered at (f, t).

A. Multiscale Moment Initialization

The receiver first forms a set of candidate channel estimates using moment matching. Since the constellation has unit average power, the amplitude follows from the received energy:

$$\widehat{|H|}_{f,t} = \sqrt{\max\big(\langle |Y|^2\rangle_{f,t} - N_0,\, 0\big)}. \tag{27}$$

The phase is recovered from the mth-order moments through the approximation $\langle Y^m\rangle_{f,t} \approx H^m_{f,t}\,\mu_m$, with $\mu_m = \frac{1}{|\mathcal{C}|}\sum_{c\in\mathcal{C}} c^m$. Taking the mth root leaves an m-fold ambiguity,

$$\widehat{\theta}^{(m,r)}_{f,t} = \frac{1}{m}\arg\left(\frac{\langle Y^m\rangle_{f,t}}{\mu_m}\right) + \frac{2\pi r}{m}, \quad r = 0, \ldots, m-1, \tag{28}$$

so that the orders $m \in \{2, 3, 4, 6\}$ used in the algorithm yield $2 + 3 + 4 + 6 = 15$ candidate phases, hence 15 candidate channels $\widehat{H}^{(m,r)}_{f,t} = \widehat{|H|}_{f,t}\, e^{j\widehat{\theta}^{(m,r)}_{f,t}}$. The moments are estimated at three time–frequency window aspect ratios for the averaging $\langle\cdot\rangle_{f,t}$ to cover different Doppler and delay profiles. At each scale, the N_{top} candidates with the highest log-likelihood,

$$S^{(m,r)}_{f,t} = \left\langle \ln\left(\sum_{c\in\mathcal{C}} \exp\left(-\frac{\left|Y_{f,t} - \widehat{H}^{(m,r)}_{f,t} c\right|^2}{N_0}\right)\right)\right\rangle_{f,t}, \tag{29}$$

are retained, leaving $3N_{\mathrm{top}}$ winners per RE. To these are added N_{top} candidates obtained by fusing the rank-matched winners of the three scales through a score-weighted average, leading to a cross-scale blended estimate where the scales agree, while the per-scale winners are retained where they disagree. The result is a pool of $K = 4N_{\mathrm{top}}$ candidates per RE, which we reindex by a single label k and denote by $\{\widehat{H}^{(k)}_{f,t}\}_{k=1}^K$, with corresponding log-likelihoods $\{S^{(k)}_{f,t}\}_{k=1}^K$.

B. Hypothesis Smoothing

Since the moment-based estimates are computed locally and the channel varies smoothly, the discrete choice among the K candidates should be coherent across the grid. This is achieved by modeling the resource grid as a conditional random field over the candidates. Let us denote by ℓ_u the random variable that assigns to each RE u the "true" channel candidate index out of the K candidates, and by $\boldsymbol{\ell} = \{\ell_u\}$ the grid-wide assignment. The posterior on $\boldsymbol{\ell}$ is modeled as

$$P(\boldsymbol{\ell} \mid \mathbf{Y}) \propto \exp\left(-\tfrac{1}{\tau}\Big(-\alpha\sum_u S^{(\ell_u)}_u + \lambda \sum_{(u,v)\in\mathcal{E}} w_{uv}\left|\widehat{H}^{(\ell_u)}_u - \widehat{H}^{(\ell_v)}_v\right|^2\Big)\right), \tag{30}$$

where α and λ are hyperparameters and $\mathcal{E}$ collects pairs of neighboring REs. The w_{uv} are uniform weights over a boxcar window in time and frequency. Since the scores $S^{(k)}_u$ are log-likelihoods, the unary term is the log-likelihood of $\boldsymbol{\ell}$ over the resource grid under an independence assumption. The pairwise term penalizes neighbors that select channels with dissimilar values.

Computing the marginals of (30) is intractable, so the posterior is approximated by a product of per-node distributions $Q(\boldsymbol{\ell}) = \prod_u q_u(\ell_u)$, with $q_u(k) \geq 0$, $\sum_{k=1}^K q_u(k) = 1$, chosen to minimize the Kullback–Leibler divergence $\mathrm{KL}\big(Q \,\|\, P(\cdot \mid \mathbf{Y})\big)$. Each factor $q_u(k)$ is then the approximate marginal probability that node u takes candidate k. Zeroing the gradient of this divergence with respect to each $q_u(k)$ yields the fixed-point update

$$q_u(k) \propto \exp\left(\tfrac{1}{\tau}\Big(\alpha\, S^{(k)}_u - \lambda\left|\widehat{H}^{(k)}_u - \bar{H}_u\right|^2\Big)\right) \tag{31}$$

for all $k \in \{1, \ldots, K\}$, in which each node sees its neighbors only through

$$\bar{H}_u = \sum_{v\in\mathcal{N}(u)} w_{uv} \sum_{k=1}^K q_v(k)\, \widehat{H}^{(k)}_v, \tag{32}$$

that is, the soft-assigned average channel of neighbors $\mathcal{N}(u)$. Note that the implementation approximates (32) by

a boxcar window centered on u, which also includes the node's own soft channel. Since $\bar{H}_u$ in (32) depends on the same assignments updated by (31), the two are applied alternately: starting from the unary-only assignment $q_u(k) \propto \exp\left(\alpha\, S_u^{(k)}/\tau\right)$, a few sweeps iterate (32) and (31).

C. Expectation-Maximization

The previous two stages aim to provide strong initializations for an EM loop that jointly refines the channel estimates and computes posterior symbol probabilities. Each of the K candidates is refined in parallel, with $\mathbf{Y}$ as the observed data, the channel $\mathbf{H}$ as the parameter to be estimated, and the transmitted symbols as the latent variables. The E-step computes soft symbol posteriors under the current channel,

$$p_{f,t}^{(k)}(c) \propto \exp\left(-\left|Y_{f,t} - \widehat{H}_{f,t}^{(k)}\, c\right|^2 / N_0\right), \quad c \in \mathcal{C}. \quad (33)$$

The M-step re-estimates the channel by treating the resulting soft symbols as pseudo-pilots, implementing a confidence-weighted least-squares fit over the local window,

$$\widehat{H}_{f,t}^{(k)} \leftarrow \frac{\langle w_{f,t}^{(k)}\, Y_{f,t}\, \widehat{c}_{f,t}^{(k)*} \rangle_{f,t}}{\langle w_{f,t}^{(k)}\, \widehat{v}_{f,t}^{(k)} \rangle_{f,t}}, \quad (34)$$

with soft moments $\widehat{c}_{f,t}^{(k)} = \sum_c p_{f,t}^{(k)}(c)\, c$, $\widehat{v}_{f,t}^{(k)} = \sum_c p_{f,t}^{(k)}(c)\, |c|^2$, and a per-candidate weight $w_{f,t}^{(k)} \propto (q_{f,t}(k))^{\kappa}\, (\max_c p_{f,t}^{(k)}(c))^{\beta}$ that combines the posterior $q_{f,t}(k)$ from the previous stage with the symbol-decision probabilities, where κ and β are hyperparameters.

After the EM loop, a probability mass function over the candidates is defined by blending the smoothed posterior with how well each candidate explains the received data,

$$\pi_{f,t}(k) \propto \left(q_{f,t}(k)\right)^{\rho} \exp\left(\tfrac{1-\rho}{\tau_\pi}\, S_{f,t}^{(k)}\right), \quad 1 \le k \le K, \quad (35)$$

such that $\sum_{k=1}^{K} \pi_{f,t}(k) = 1$, where $S_{f,t}^{(k)}$ is the local mixture log-likelihood of the form (29) re-evaluated at the converged channel $\widehat{H}_{f,t}^{(k)}$, $q_{f,t}(k)$ is the mean-field posterior obtained by reapplying the smoothing of (31) to these refined scores, and $\rho \in [0,1]$ and τ_π are hyperparameters. These weights yield a fused posterior on the transmitted symbols

$$\bar{p}_{f,t}(c) = \sum_{k=1}^{K} \pi_{f,t}(k)\, p_{f,t}^{(k)}(c), \quad c \in \mathcal{C}. \quad (36)$$

The loop also returns a per-RE channel-estimation variance $\sigma_{H,f,t}^2$, used later for computing the LLRs.

D. Ratio-Prior Belief Propagation

While (36) gives probability mass functions for all symbols in the resource grid, the true posteriors are coupled, because the channel is approximately constant over neighboring REs. Hence, more accurate marginals of the grid-wide posterior can be obtained through a belief propagation-inspired scheme on a Markov random field. This first refinement approximates, for each RE, the marginal of its symbol under the grid-wide posterior,

$$P(c_{f,t} \mid \mathbf{Y}) = \sum_{\boldsymbol{c} \setminus c_{f,t}} P(\boldsymbol{c} \mid \mathbf{Y}), \quad (37)$$

where $\boldsymbol{c} = \{c_{f,t}\}$ is the grid-wide symbol labeling, and turns it into a per-RE symbol log-domain refinement $\Gamma_{f,t}(c)$ that is fed back into the estimation. As all the REs are coupled through the unknown channel, (37) is intractable.

It is assumed that the dependency between symbols is restricted to time- and frequency-adjacent REs, which makes $P(\boldsymbol{c} \mid \mathbf{Y})$ a pairwise Markov random field on the resource grid. For such a field, the Hammersley–Clifford theorem gives the factorization into unary and pairwise potentials,

$$P(\boldsymbol{c} \mid \mathbf{Y}) \;\propto\; \prod_u \psi_u(c_u) \prod_{(u,v)\in\mathcal{E}} \psi_{uv}(c_u, c_v), \quad (38)$$

where u and v index REs and $\mathcal{E}$ is the set of neighboring RE pairs. The two potentials are set as follows. The unary potential is set to $\psi_u(c) = \bar{p}_u(c)^{\,w_{\mathrm{src}}}$, where w_{src} is a hyperparameter. The pairwise potential scores adjacent symbols through the ratio of their received samples: two adjacent REs are assumed to experience approximately the same channel ($H_u \approx H_v$), so the channel approximately cancels in that ratio,

$$\frac{Y_u}{Y_v} = \frac{H_u X_u + W_u}{H_v X_v + W_v} \approx \frac{X_u}{X_v}, \quad (39)$$

and the observed ratio $q_{uv} = Y_u / Y_v$ depends on the two symbols only. The corresponding potential is

$$\psi_{uv}(c_u, c_v) = e^{\phi_{uv}(c_u, c_v)}, \text{ with}$$
$$\phi_{uv}(c_u, c_v) = -g\, \ln\left(1 + \frac{|q_{uv} - c_u/c_v|^2}{\sigma_{uv}^2}\right), \quad (40)$$

with g a hyperparameter and ratio-error variance $\sigma_{uv}^2 \propto N_0(1 + |q_{uv}|^2)/\,|Y_v|^2$. This factor has heavy Cauchy-like tails, which suppress edges near a deep fade where $|Y_v|$ is small and the ratio is unreliable.

The marginals (37) of the field (38) are approximated by a belief propagation-inspired approach, with each symbol restricted to the set $\mathcal{M}_{f,t} \subset \mathcal{C}$ of its most likely values under the EM posterior. The edge messages are

$$\mu_{v\to u}(c) \propto \sum_{c' \in \mathcal{M}_v} \exp\left(\phi_{uv}(c, c') + w_{\mathrm{bp}}\, b_v(c')\right), \quad (41)$$

with w_{bp} a hyperparameter. The node belief $b_u(c)$ combines the incoming log-messages with the unary log-potential,

$$b_u(c) = \frac{1}{d_u} \sum_{v \in \mathcal{N}(u)} w_{uv}\, \ln\left(\mu_{v\to u}(c)\right) + w_{\mathrm{src}}\, \mathcal{R}_u\, \ln\left(\bar{p}_u(c)\right), \quad (42)$$

where $\mathcal{N}(u)$ is the set of neighbors of u and $d_u = \sum_{v\in\mathcal{N}(u)} w_{uv}$ is the weighted node degree. Here $\mathcal{R}_u \in [0,1]$ is a per-node reliability and w_{uv} an edge weight, whose detailed definitions are omitted. While the messages and

beliefs could be updated alternately over the grid, the implementation runs a single sweep, comprising one frequency- and one time-adjacent pass.

The resulting beliefs, mapped back onto the constellation, give the per-RE symbol refinement $\Gamma_{f,t}(c)$. This refinement is added as a symbol log-prior in a further EM pass, which re-estimates the channel $\widehat{H}_{f,t}^{(k)}$ and the symbol posteriors using the following E-step in place of (33):

$$p_{f,t}^{(k)}(c) \propto \exp\left(- \left| Y_{f,t} - \widehat{H}_{f,t}^{(k)}\, c \right|^2 / N_0 + w'_\Gamma\, \Gamma_{f,t}(c) \right), \tag{43}$$

where w'_Γ is a hyperparameter.

E. Decision-Directed Tracking and Fusion

The second refinement consists in using the fused soft symbol estimate $\widehat{c}_{f,t} = \sum_{c\in\mathcal{C}} \bar{p}_{f,t}(c)\, c$ as a virtual pilot to form a decision-directed estimate,

$$\widehat{H}_{f,t}^{\mathrm{ddtf}} = \frac{Y_{f,t}\, \widehat{c}_{f,t}^{\,*}}{\widehat{E}_{f,t}}, \qquad \widehat{E}_{f,t} = \sum_{c\in\mathcal{C}} \bar{p}_{f,t}(c)\, |c|^2 . \tag{44}$$

Each RE is assigned a reliability

$$\eta_{f,t} = \chi_{f,t}\, \frac{|H^\star_{f,t}|^2}{|H^\star_{f,t}|^2 + \sigma^2_{\mathrm{ddtf},f,t}} \tag{45}$$

with

$$\sigma^2_{\mathrm{ddtf},f,t} = \frac{N_0 + |H^\star_{f,t}|^2\, \mathrm{Var}_{f,t}(c)}{\widehat{E}_{f,t}}, \tag{46}$$

where $H^\star_{f,t} = \sum_k \pi_{f,t}(k)\, \widehat{H}_{f,t}^{(k)}$ is the trust-weighted mean channel, $\mathrm{Var}_{f,t}(c) = \widehat{E}_{f,t} - |\widehat{c}_{f,t}|^2$ is the soft-symbol variance, and $\chi_{f,t} \in [0,1]$ is a symbol-confidence factor whose computation details are omitted. The estimate is smoothed by a reliability-weighted Gaussian convolution,

$$\widehat{H}_{f,t}^{\mathrm{ddtf}} \leftarrow \frac{\sum_{f',t'} \mathcal{G}_{f-f',\,t-t'}\, \eta_{f',t'}\, \widehat{H}_{f',t'}^{\mathrm{ddtf}}}{\sum_{f',t'} \mathcal{G}_{f-f',\,t-t'}\, \eta_{f',t'}}, \tag{47}$$

with $\mathcal{G}$ a fixed Gaussian kernel, so that confidently estimated REs dominate and unreliable ones are filled in from their neighbors. The same convolution returns a channel-estimation variance $\sigma^2_{H,f,t}$ from the smoothed σ^2_{ddtf}.

The smoothed track is appended to the pool as an additional candidate, with its own variance $\sigma^2_{H,f,t}$. All $K+1$ candidates are then re-scored to update the trust weights $\pi_{f,t}(k)$ by the same construction as (35). Each candidate is rated by its local mixture log-likelihood of the form (29), except that the noise is inflated by the channel estimation variance, i.e., $N_0 + |c|^2\, \sigma^2_{H,f,t}$ is used instead of N_0.

F. Local Affine Correction

The third refinement denoises the channel within each small time–frequency patch by fitting an affine surface

$$H(f+\delta f,\, t+\delta t) \approx p_0 + p_1 \delta t + p_2 \delta f = \mathbf{d}^\top \mathbf{p} \tag{48}$$

over a 5×3 window $\delta t \in \{-2,\dots,2\}$, $\delta f \in \{-1,0,1\}$ centered on (f,t), where $\mathbf{d} = [1, \delta t, \delta f]^\top$ and $\mathbf{p} = [p_0, p_1, p_2]^\top \in \mathbb{C}^3$, so that p_0 is the channel at the patch center and p_1, p_2 are its temporal and spectral gradients. The parameters minimize the following objective over the patch,

$$J(\mathbf{p}) = \sum_{\delta f,\delta t} w_{f',t'}\, \mathbb{E}\big[\, \left| Y_{f',t'} - (\mathbf{d}^\top \mathbf{p})\, c_{f',t'} \right|^2 \big] + \lambda_0 \left| p_0 - H^\star_{f,t} \right|^2 + \lambda_s \left(|p_1|^2 + |p_2|^2 \right), \tag{49}$$

where $(f',t') = (f+\delta f, t+\delta t)$ and the expectation is taken over the fused soft symbol at each patch RE, yielding $\mathbb{E}[\,|Y - (\mathbf{d}^\top\mathbf{p})c|^2\,] = |Y - (\mathbf{d}^\top\mathbf{p})\widehat{c}|^2 + |\mathbf{d}^\top\mathbf{p}|^2\, \mathrm{Var}(c)$. The weight $w_{f',t'} = \max(\max_{c\in\mathcal{C}} \bar{p}_{f',t'}(c),\, w_0)^a$ grows with the demapping confidence of the RE, so that reliably demapped REs contribute most to the fit. The regularization anchors p_0 to the trust-weighted prior $H^\star_{f,t} = \sum_k \pi_{f,t}(k)\, \widehat{H}_{f,t}^{(k)}$ with strength λ_0 and shrinks the slopes with strength λ_s. Being quadratic in $\mathbf{p}$, (49) is solved in closed form,

$$\mathbf{p} = \Big(\textstyle\sum_{\delta f,\delta t} w_{f',t'}\, \widehat{E}_{f',t'}\, \mathbf{d}\mathbf{d}^\top + \mathbf{R} \Big)^{-1} \cdot \Big(\textstyle\sum_{\delta f,\delta t} w_{f',t'}\, \widehat{c}^{\,*}_{f',t'}\, Y_{f',t'}\, \mathbf{d} + \mathbf{r}_0 \Big), \tag{50}$$

with $\mathbf{R} = \mathrm{diag}(\lambda_0, \lambda_s, \lambda_s)$ and $\mathbf{r}_0 = [\lambda_0 H^\star_{f,t}, 0, 0]^\top$. Here, $\widehat{E} = |\widehat{c}|^2 + \mathrm{Var}(c)$. The center value p_0 is blended with its 3×3 neighborhood for spatial consistency, then replaces the dominant candidate through a soft gate that opens according to a confidence threshold. The channel-estimation variance $\sigma^2_{H,f,t}$ is correspondingly inflated by the patch residual and the magnitude of the applied correction.

G. Demapper

Finally, the bit LLRs are computed by a demapper that sums the likelihood over both the constellation points and the competing channel candidates. It first forms a per-symbol log-metric in which the noise variance of each candidate is inflated by its own channel-estimation variance. The K candidates refined by the EM loop share the common variance $\sigma^2_{H,f,t}$, whereas the appended decision-directed track carries its own variance and the affine correction inflates the variance of the candidate it replaces. Together these define the per-candidate variance $\sigma^{2,(k)}_{H,f,t}$. Let $\nu^{(k)}_{f,t}(c) \triangleq N_0 + |c|^2\, \sigma^{2,(k)}_{H,f,t}$ be the inflated noise variance. Then

$$\Lambda_{f,t}(c) = \ln\left(\sum_k \frac{\pi_{f,t}(k)}{\nu^{(k)}_{f,t}(c)}\, \exp\left(- \frac{\left| Y_{f,t} - \widehat{H}_{f,t}^{(k)}\, c \right|^2}{\nu^{(k)}_{f,t}(c)} \right) \right) + w_\Gamma\, \Gamma_{f,t}(c). \tag{51}$$

The ratio refinement $\Gamma_{f,t}$ is added as a log-prior of weight w_Γ. Because the inflated variance also depends on the symbol through $|c|^2$, this normalizer is symbol-dependent and penalizes the higher-energy symbols. The raw LLR of bit b marginalizes this metric over the subsets $\mathcal{C}_{b,1}$ and $\mathcal{C}_{b,0}$ of constellation points whose bth bit equals 1 and 0, respectively,

$$\mathrm{LLR}^{\mathrm{raw}}_{f,t,b} = \ln\left(\frac{\sum_{c\in\mathcal{C}_{b,1}} e^{\Lambda_{f,t}(c)}}{\sum_{c\in\mathcal{C}_{b,0}} e^{\Lambda_{f,t}(c)}} \right). \tag{52}$$

Since $\Gamma_{f,t}$ has already been injected upstream into the channel estimates and trust weights, and through (51) into the metric itself, the raw LLR is not extrinsic to the prior. To provide an approximation of the demapper's extrinsic information, the receiver subtracts a reliability-weighted fraction of the prior's standalone bit-LLR $L^{\Gamma}_{f,t,b}$,

$$\mathrm{LLR}_{f,t,b} = \mathrm{LLR}^{\mathrm{raw}}_{f,t,b} - \lambda_{f,t}\, L^{\Gamma}_{f,t,b}, \tag{53}$$

where the damping $\lambda_{f,t}$ grows where $\Gamma_{f,t}$ is less reliable, so an uncertain prior is suppressed more strongly in the output.

Acknowledgment

The candidate algorithm code discussed in Sections III–IV and Appendixes B–C was generated by GPT-5.5 within AITE; the authors verified the implementations and edited the accompanying descriptions. OpenAI Codex with GPT-5.5 was used to check for grammar and spelling errors.

REFERENCES

[1] F. Aït Aoudia, J. Hoydis, S. Cammerer, L. Maggi, G. Marti, and A. Keller, "The AI Telco Engineer: Toward autonomous discovery of wireless communications algorithms," in *European Wireless*, Rimini, Italy, Jun. 2026. [Online]. Available: https://arxiv.org/abs/2604.19803

[2] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[3] J. Lehman, J. Gordon, S. Jain, K. Ndousse, C. Yeh, and K. O. Stanley, "Evolution through large models," in *Handbook of Evolutionary Machine Learning*. Springer Nature Singapore, 2024, pp. 331–366. [Online]. Available: https://arxiv.org/abs/2206.08896

[4] E. Meyerson *et al.*, "Language model crossover: Variation through few-shot prompting," *ACM Trans. Evol. Learn. Optim.*, vol. 4, no. 4, pp. 1–40, Nov. 2024. [Online]. Available: https://doi.org/10.1145/3694791

[5] A. Chen, D. M. Dohan, and D. R. So, "EvoPrompting: Language models for code-level neural architecture search," in *Proc. 37th Conf. Neural Inf. Process. Syst. (NeurIPS)*, New Orleans, LA, USA, Dec. 2023, pp. 7787–7817.

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

[7] A. Novikov *et al.*, "AlphaEvolve: A coding agent for scientific and algorithmic discovery," *arXiv preprint arXiv:2506.13131*, Jun. 2025. [Online]. Available: https://arxiv.org/abs/2506.13131

[8] A. Sharma, "OpenEvolve: An open-source evolutionary coding agent," https://github.com/algorithmicsuperintelligence/openevolve, 2025.

[9] R. T. Lange, Y. Imajuku, and E. Cetin, "ShinkaEvolve: Towards open-ended and sample-efficient program evolution," *arXiv preprint arXiv:2509.19349*, 2025. [Online]. Available: https://arxiv.org/abs/2509.19349

[10] L. A. Agrawal *et al.*, "GEPA: Reflective prompt evolution can outperform reinforcement learning," in *Int. Conf. Learn. Representations (ICLR)*, Rio de Janeiro, Brazil, Apr. 2026. [Online]. Available: https://openreview.net/forum?id=RQm2KQTM5r

[11] S. Liu *et al.*, "SkyDiscover: A flexible, adaptive framework for AI-driven scientific and algorithmic discovery," in *Proc. ACM Conf. AI Agentic Syst. (CAIS'26)*, San Jose, CA, USA, May 2026, pp. 1223–1227.

[12] M. Cemri *et al.*, "AdaEvolve: Adaptive LLM driven zeroth-order optimization," *arXiv preprint arXiv:2602.20133*, Feb. 2026. [Online]. Available: https://arxiv.org/abs/2602.20133

[13] S. Liu *et al.*, "EvoX: Meta-evolution for automated discovery," *arXiv preprint arXiv:2602.23413*, Feb. 2026. [Online]. Available: https://arxiv.org/abs/2602.23413

[14] T. Chen *et al.*, "AVO: Agentic variation operators for autonomous evolutionary search," *arXiv preprint arXiv:2603.24517*, Mar. 2026. [Online]. Available: https://arxiv.org/abs/2603.24517

[15] F. Weindel and R. Heckel, "LLM-guided search for deletion-correcting codes," *Trans. Mach. Learn. Res.*, Jun. 2026. [Online]. Available: https://openreview.net/forum?id=qZ69Ozpo6v

[16] G. Zheng, F. Liu, and Q. Zhang, "LLM-enabled automated algorithm design for multiuser fluid antenna communications," *IEEE Trans. Wireless Commun.*, May 2026, to appear. [Online]. Available: https://arxiv.org/abs/2605.14661

[17] T. Aghayev *et al.*, "GENESIS: Harnessing AI agents for autonomous 6G RAN synthesis, research, and testing," *arXiv preprint arXiv:2605.27360*, May 2026. [Online]. Available: https://arxiv.org/abs/2605.27360

[18] F. Aït Aoudia, J. Hoydis, S. Cammerer, L. Maggi, G. Marti, and A. Keller, "The AI Telco Engineer," https://github.com/nvlabs/the-ai-telco-engineer, 2026.

[19] LangChain, Inc., "LangChain," https://github.com/langchain-ai/langchain, 2022, version 1.3.11.

[20] S. K. Mohammed, R. Hadani, A. Chockalingam, and R. Calderbank, "OTFS—a mathematical foundation for communication and radar sensing in the delay-Doppler domain," *IEEE BITS Inf. Theory Mag.*, vol. 2, no. 2, pp. 36–55, Nov. 2022.

[21] F. Aït Aoudia and J. Hoydis, "End-to-end learning for OFDM: From neural receivers to pilotless communication," *IEEE Trans. Wireless Commun.*, vol. 21, no. 2, pp. 1049–1063, Feb. 2022.

[22] S. Yao *et al.*, "ReAct: Synergizing reasoning and acting in language models," in *Int. Conf. Learn. Representations (ICLR)*, Kigali, Rwanda, May 2023. [Online]. Available: https://arxiv.org/abs/2210.03629

[23] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, Anchorage, AK, USA, Aug. 2019, pp. 2623–2631.

[24] J. Hoydis *et al.*, "Sionna," https://nvlabs.github.io/sionna/, 2022, version 2.0.1.

[25] J. Ansel *et al.*, "PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation," in *ACM Int. Conf. Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, La Jolla, CA, USA, May 2024, pp. 929–947. [Online]. Available: https://doi.org/10.1145/3620665.3640366

[26] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, Jul. 2021.

[27] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *Proc. 51st Annu. Conf. Inf. Sci. Syst. (CISS)*, Baltimore, MD, USA, May 2017, pp. 1–6.

[28] P. Raviteja, K. T. Phan, Y. Hong, and E. Viterbo, "Interference cancellation and iterative detection for orthogonal time frequency space modulation," *IEEE Trans. Wireless Commun.*, vol. 17, no. 10, pp. 6501–6515, Oct. 2018.

[29] Z. Yuan, F. Liu, W. Yuan, Q. Guo, Z. Wang, and J. Yuan, "Iterative detection for orthogonal time frequency space modulation with unitary approximate message passing," *IEEE Trans. Wireless Commun.*, vol. 21, no. 2, pp. 714–725, Feb. 2022.

[30] H. Li, Y. Dong, C. Gong, Z. Zhang, X. Wang, and X. Dai, "Low complexity receiver via expectation propagation for OTFS modulation," *IEEE Commun. Lett.*, vol. 25, no. 10, pp. 3180–3184, Oct. 2021.

[31] S. Tiwari, S. S. Das, and V. Rangamgari, "Low complexity LMMSE receiver for OTFS," *IEEE Commun. Lett.*, vol. 23, no. 12, pp. 2205–2209, Dec. 2019.

[32] G. D. Surabhi and A. Chockalingam, "Low-complexity linear equalization for OTFS modulation," *IEEE Commun. Lett.*, vol. 24, no. 2, pp. 330–334, Feb. 2020.

[33] S. Rangan, P. Schniter, and A. K. Fletcher, "Vector approximate message passing," *IEEE Trans. Inf. Theory*, vol. 65, no. 10, pp. 6664–6684, Oct. 2019.

[34] J. Ma and L. Ping, "Orthogonal AMP," *IEEE Access*, vol. 5, pp. 2020–2033, Jan. 2017.

[35] M. Opper and O. Winther, "Expectation consistent approximate inference," *J. Mach. Learn. Res.*, vol. 6, no. 73, pp. 2177–2204, Dec. 2005. [Online]. Available: https://jmlr.org/papers/v6/opper05a.html

[36] P. L. Combettes and J.-C. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*. New York, NY, USA: Springer, 2011, pp. 185–212.

[37] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction

method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.
[38] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. Expected complexity," *IEEE Trans. Signal Process.*, vol. 53, no. 8, pp. 2806–2818, Aug. 2005.
[39] K. W. Wong, C. Y. Tsui, R. S. K. Cheng, and W. H. Mow, "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Scottsdale, AZ, USA, May 2002, pp. III-273–III-276.
[40] S. Rangan, "Generalized approximate message passing for estimation with random linear mixing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, St. Petersburg, Russia, Aug. 2011, pp. 2168–2172.
[41] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proc. Nat. Acad. Sci.*, vol. 106, no. 45, pp. 18 914–18 919, Nov. 2009.
[42] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: SIAM, 2003.
[43] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, no. 1, pp. 79–87, Mar. 1991.
[44] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: Johns Hopkins Univ. Press, 2013.
[45] M.-X. Chang and J.-H. Chen, "Efficient blind detection for OFDM systems in selective fading channels," *IEEE Trans. Veh. Technol.*, vol. 64, no. 5, pp. 1779–1792, May 2015.
[46] T. Y. Al-Naffouri, A. A. Dahman, M. S. Sohail, W. Xu, and B. Hassibi, "Low-complexity blind equalization for OFDM systems with general constellations," *IEEE Trans. Signal Process.*, vol. 60, no. 12, pp. 6395–6407, Dec. 2012.
[47] S. A. Banani and R. G. Vaughan, "OFDM with iterative blind channel estimation," *IEEE Trans. Veh. Technol.*, vol. 59, no. 9, pp. 4298–4308, Nov. 2010.
[48] L. Bariah, A. Al-Dweik, and S. Muhaidat, "Blind channel estimation technique for OFDM systems over time varying channels," in *Proc. IEEE 87th Veh. Technol. Conf. (VTC Spring)*, Porto, Portugal, Jul. 2018, pp. 1–7.
[49] A. Saci, A. Al-Dweik, A. Shami, and Y. Iraqi, "One-shot blind channel estimation for OFDM systems over frequency-selective fading channels," *IEEE Trans. Commun.*, vol. 65, no. 12, pp. 5445–5458, Dec. 2017.
[50] E. Björnson, M. Dohler, J. Hoydis, and R. W. Heath, Jr., "Automating wireless research and development: What remains human?" *Preprints*, Apr. 2026. [Online]. Available: https://doi.org/10.20944/preprints202604.1172.v1