# OmniQEC: discovering practical quantum error-correcting codes by an AI scientist

Ge Yan,[1] Shanchuan Li,[1,2] Pengyue Ma,[1] Qixin Zhang,[1] Pingchuan Ma,[3] Jianping Wang,[4] Min-Hsiu Hsieh,[5] and Yuxuan Du[1,6,*]

[1] *College of Computing and Data Science, Nanyang Technological University, Singapore*
[2] *Department of Electrical Engineering and Computer Science, Tokyo University of Agriculture and Technology, Koganei, Tokyo, Japan*
[3] *Zhejiang University of Technology, China*
[4] *Department of Computer Science, City University of Hong Kong, Hong Kong, China*
[5] *Hon Hai (Foxconn) Research Institute, Taipei, Taiwan*
[6] *School of Physical and Mathematical Science, Nanyang Technological University, Singapore*

Quantum error correction (QEC) is indispensable for scalable fault-tolerant quantum computing. However, discovering QEC codes that remain effective is challenging, as logical performance depends on the interplay between code structure, hardware, syndrome extraction, and decoding, which often impose competing requirements. Here we introduce OmniQEC, an efficient AI scientist for discovering QEC codes suited to deployment on modern quantum processors. OmniQEC formulates QEC design as an iterative discovery process in which an orchestrator, implemented by advanced large language models (LLMs), coordinates code generation, code-level screening, syndrome-extraction synthesis, and decoder-based circuit evaluation. At its core, OmniQEC combines a self-evolving reasoning mechanism with a slow–fast synergistic workflow: a fast loop explores candidates using inexpensive code-level proxies, whereas a slow loop performs physically grounded circuit-level evaluation and feeds the resulting evidence back into the search. We evaluate OmniQEC across four qLDPC construction families, three LLM backends, and 14 total-physical-qubit budgets per backend. The discovered codes show steadily improving logical-error suppression with increasing physical-qubit budgets and outperform the BB codes with $[\![72, 12, 6]\!]$ and $[\![144, 12, 12]\!]$ under complete-implementation budgets of 98 and 240 physical qubits, respectively. The discovered codes are hardware-friendly and may be of independent interest for practical QEC implementation. These findings pave the way towards LLM-assisted QEC discovery grounded in physically informed code–circuit–decoder co-design.



## I. INTRODUCTION

Quantum error correction (QEC) is the foundation for fault-tolerant quantum computation (FTQC) [1–3]. Its central role has driven a broad landscape of code families. Representative codes include topological surface and toric codes for geometrically local protection [4–6], color codes with transversal operations [7, 8], subsystem and Floquet codes for simplified syndrome extraction [9–12], high-rate qLDPC codes for low-overhead memory with sparse checks and flexible connectivity [13–17], and concatenated or bosonic codes for hierarchical or mode-level protection [18, 19]. Despite this diversity, only a small subset of these constructions has reached modern quantum processors, including surface-code experiments [20, 21] and emerging colour-code logic [8] on superconducting devices, encoded logical circuits on reconfigurable atom arrays [22, 23], and real-time fault-tolerant logical operations with trapped ions [24, 25]. This gap reflects a conventional code-centric paradigm that separates algebraic construction from the constraints of practical deployment [26]. As hardware platforms progress towards early FTQC [27], flexible architectures expand the experimentally accessible qLDPC design space [28]. This expanded design space, however, places practical and high-performing QEC codes increasingly beyond the reach of purely expert-driven search [5, 14, 16, 17].

Recent advances in artificial intelligence (AI) have initiated early attempts to automate the discovery of practical QEC codes. Existing approaches broadly fall into three categories. One class uses reinforcement learning to search over code constructions with reward functions defined by favorable properties [29–34]. A second uses Bayesian optimization to guide candidate selection [35], whereas a third builds on large language model (LLM)-guided program search to generate and iteratively refine executable code constructions [36–38]. Nevertheless, all paradigms remain predominantly *code-centric*, ranking candidates using algebraic quantities such as $[\![n, k, d]\!]$ or inexpensive proxies derived from them [39–41]. In other words, they fail to capture the practical determinants of QEC performance, such as hardware constraints, syndrome extraction, and decoding. As a result, the designed QEC code that appears promising at the code level may deliver poor logical performance in practice [42–44]. This raises a central question:

*how can QEC discovery move beyond code-level optimization towards practical performance?*

* yuxuan.du@ntu.edu.sg

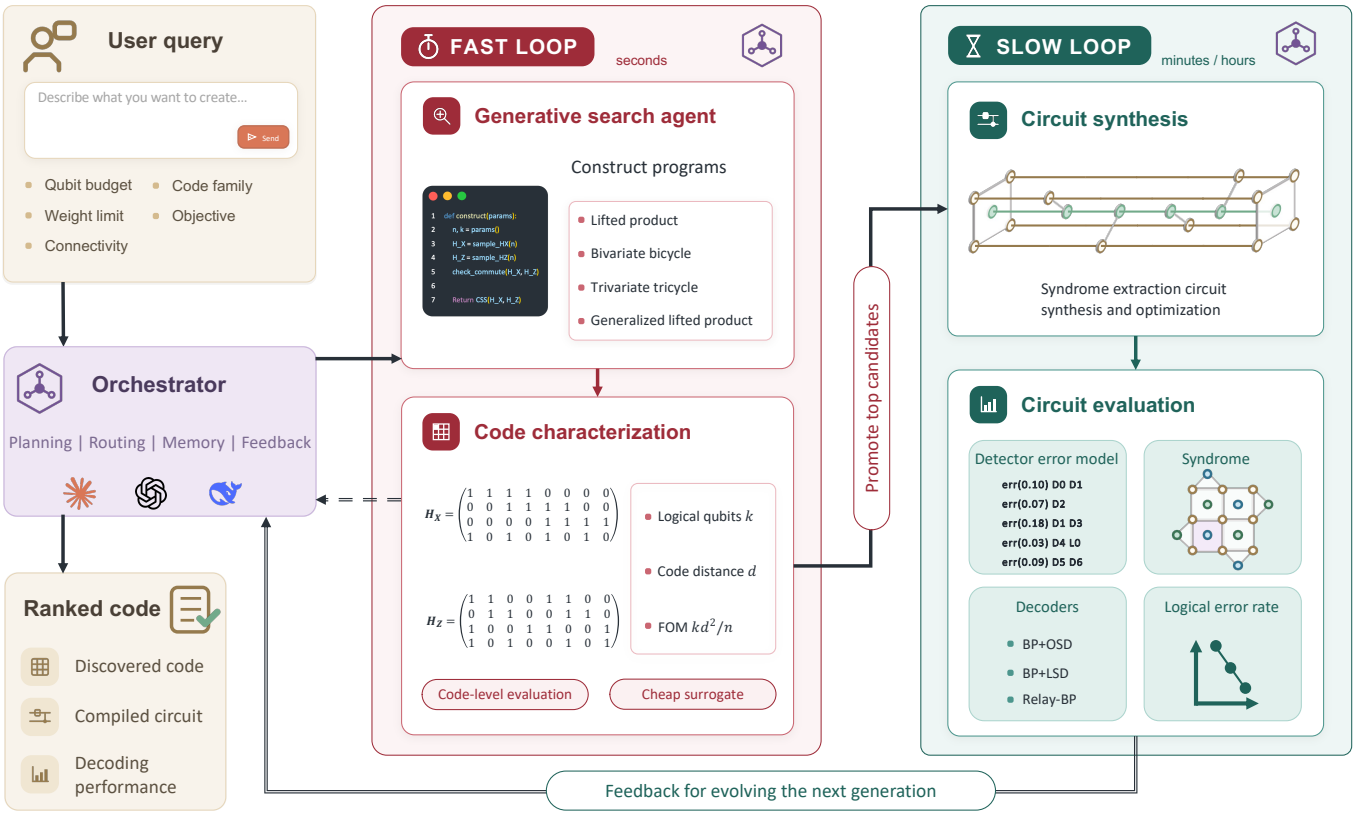


FIG. 1. **Schematic illustration of OmniQEC.** A user specifies a QEC design objective, such as the target code family, optimization criterion, physical-qubit budget, connectivity, native gate set, and available ancilla resources. The LLM-based orchestrator translates this objective into a structured discovery task, maintains the self-evolving discovery memory, and routes information across a dual-loop workflow. In the fast loop, the code-search agent proposes executable code-construction programs, which are converted into candidate QEC codes and screened by the code-characterization module using code-level properties, including validity, logical-qubit number, code distance, and the code-level proxy such as figure of merit (FOM). Promising candidates are promoted to the slow loop, where the circuit-synthesis module compiles validated codes into syndrome-extraction circuits and the circuit-evaluation module estimates circuit-level logical error rates using noisy sampling and selected decoders. The resulting circuit-level evidence is returned to the orchestrator as slow-loop feedback and incorporated into the discovery memory, enabling subsequent iterations to generate improved candidates. The workflow outputs a ranked package containing reproducible code-construction programs, discovered codes, compiled circuits, and decoding performance.

Here we introduce OmniQEC, an AI scientist for discovering practical QEC codes by directly optimizing the circuit-level logical error rate (LER), a standard metric of implementation-level QEC performance [20, 21]. Conceptually, OmniQEC casts QEC code discovery as an autonomous generate–evaluate–refine process, in which an LLM-based orchestrator harnesses a *self-evolving reasoning mechanism* to propose candidate constructions, invoke purpose-built scientific tools for validation, and refine subsequent search strategies using the resulting evidence. In addition, to reconcile the large search space with the high cost of LER evaluation, OmniQEC exploits a *slow–fast synergistic workflow*. The fast loop generates and screens large populations of candidates using inexpensive code-level metrics, whereas the slow loop compiles selected candidates into noisy circuits, evaluates their LERs using configurable decoders, and feeds the resulting evidence back into both candidate generation and search refinement. In doing so, OmniQEC incorporates improved physical models and evaluation methods without restructuring the discovery loop. Together, these components deliver an efficient, self-evolving AI scientist for physically grounded code–circuit–decoder co-design.

We systematically evaluate OmniQEC under the practical constraints of deploying qLDPC codes on modern quantum processors. Specifically, we impose a fixed total physical-qubit budget $N$ and optimize the circuit-level LER over complete code implementations. Across four qLDPC construction families and three LLM backends (i.e., Claude, GPT, and DeepSeek), the scaling analysis shows that OmniQEC discovers codes with progressively stronger error suppression as $N$ increases, implying an approximately linear resource–performance trend. The discovered codes outperform the standard bivariate-bicycle (BB) code with $[\![72, 12, 6]\!]$ when $N = 98$ [17], while the strongest Claude Opus 4.8 campaign surpasses the BB benchmark with $[\![144, 12, 12]\!]$ at $N = 240$. Besides, the achieved results reveal

that the widely used code-level figure of merit $\Phi_{\text{code}} = kd^2/n$ [17, 26, 36, 45] does not always align with the circuit-level LER ranking. This discrepancy suggests a fundamental limitation of automated approaches that optimize code-level objectives alone. Further evaluations show that the discovered codes retain their performance advantage across physical error rates beyond the single value used during search. These results establish OmniQEC as a physically grounded framework for discovering qLDPC codes that admit resource-efficient processor-level implementations.

## II. IMPLEMENTATION OF OMNIQEC

Practical QEC discovery centers on bridging the gap between where candidates are generated and where their performance is ultimately determined [28]. This remains a long-standing challenge, because physical feedback is often delayed, expensive, and contingent on hardware-specific circuit realizations [46–50], leaving conventional code-centric search poorly aligned with the final objective. Progress towards addressing this challenge could advance fault-tolerant quantum computing by accelerating the identification of practically competitive architectures.

### A. OmniQEC system architecture

OmniQEC is designed to bridge this gap by integrating code discovery, syndrome-extraction circuit synthesis, and decoder-based evaluation within a unified workflow. Following the broader paradigm of AI-scientist systems [51–56], it leverages the reasoning capabilities of LLMs and the tool-use capabilities of agentic systems to identify practical QEC codes under user-specified objectives [57]. At its core, an LLM-based orchestrator interprets the user's objective, formulates a structured discovery task, and coordinates agents and computational tools throughout the discovery process. To support this discovery, we develop a suite of QEC-specific agents and tools for evaluating code properties, validating circuit-level performance, and exploring structured code families at scale.

An overview of OmniQEC is shown in Fig. 1. The workflow begins by parsing a user-defined design objective. The LLM-based orchestrator interprets the query and translates it into a structured QEC discovery task, e.g., the target code family, optimization criteria, and relevant hardware constraints. These constraints may specify the total physical-qubit budget $N$, qubit connectivity, native gate set, available ancilla resources and other implementation requirements. After the interpretation, OmniQEC initiates a self-evolving discovery process over $T$ iterations. At iteration $t \in [T]$, candidate code-construction programs are generated, evaluated, and refined by the orchestrator using feedback from previous iterations. The discovery process terminates when the target objective is met, the iteration limit $T$ is reached, or the available computational budget is exhausted.

*Dual-loop strategy.* A computational bottleneck in this iterative process is circuit-level evaluation, which requires constructing noisy syndrome-extraction circuits and estimating the corresponding logical performance through a large number of decoding simulations [58]. To address this bottleneck, we devise a *dual-loop strategy* within each iteration, as shown in the middle and right panels of Fig. 1. OmniQEC uses efficiently computed code-level properties as proxies to explore and refine a broad set of candidate constructions, while selectively allocating costly circuit-level evaluations to the most promising ones. The resulting circuit-level evidence, together with the associated reasoning trajectories [59–61], is then fed back to the orchestrator, enabling it to focus subsequent iterations on promising regions of the design space. In what follows, we present the implementation of this dual-loop discovery process and the agents and tools that support it. Further details are provided in Supplementary Information (SI) A and B.

The dual-loop strategy at each iteration contains a fast loop and a slow loop. In the fast loop, OmniQEC employs a code-search agent to propose executable construction programs for QEC codes with parameters $[\![n, k, d]\!]$ across the specified code families, such as lifted-product [62], BB [17], and multivariate-bicycle codes [63]. Each program is assessed by the code-characterization tools, which verify the validity of the resulting code, and compute code-level properties such as the number of logical qubits $k$, code distance $d$, and the selected figure of merit $\Phi_{\text{code}} = kd^2/n$ [17, 36]. For Calderbank–Shor–Steane (CSS) codes [39, 40], specified by the binary $X$- and $Z$-check matrices $H_X$ and $H_Z$, the validity includes verifying the commutation condition $H_X H_Z^{\mathsf{T}} = 0$ over $\mathbb{F}_2$ [41]. These evaluations take only seconds per candidate, allowing a large set of constructions to be rapidly evaluated and ranked. The orchestrator selects the top-ranked candidates for the slow loop.

In the slow loop, each prioritized code from the fast loop is first passed to the circuit-synthesis module, which constructs and optimizes a syndrome-extraction circuit under the specified connectivity, native gate set, and scheduling constraints. The resulting circuit is then supplied to the circuit-evaluation module, which derives the corresponding detector error model [58], generates noisy syndrome samples, and applies the selected decoders to estimate the circuit-level logical error rate (LER) [6]. This stage requires repeated noisy sampling and decoding [58, 64], which typically takes minutes to hours and is therefore reserved for a selected subset of candidates. The resulting circuit-level ranking, together with the associated synthesis, decoding, and reasoning records, is fed back to the orchestrator and

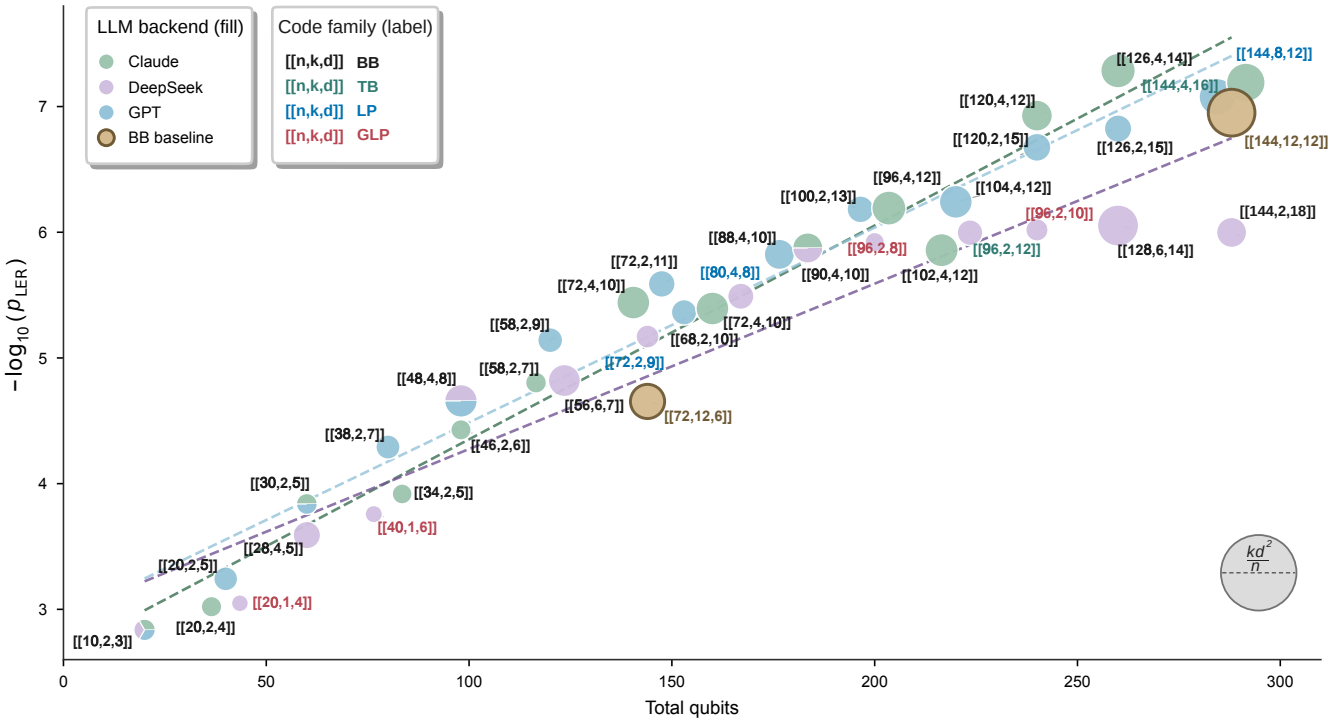


FIG. 2. **Circuit-level performance frontier discovered by OmniQEC.** Circuit-level performance as a function of the physical-qubit budget $N$, including data qubits and stabilizer-measurement ancillas. The vertical axis reports logical-error suppression, $-\log_{10}(p_{\mathrm{LER}})$, where $p_{\mathrm{LER}}$ is the logical error rate per round per logical qubit. Marker fill denotes the LLM backend used by the orchestrator, and pie markers indicate codes independently discovered by multiple backends. The BB $[\![72, 12, 6]\!]$ and $[\![144, 12, 12]\!]$ codes are included as reference codes. Label color denotes the code family, and circle area is proportional to the fast-loop code-level proxy $\Phi_{\mathrm{code}}$. The discovered codes form a clear resource–performance frontier, with several candidates achieving stronger circuit-level suppression than the BB reference codes despite having smaller $\Phi_{\mathrm{code}}$. This mismatch shows that code-level proxies can mis-rank operational performance and motivates circuit-level LER as the direct discovery objective.

incorporated into the discovery memory. The orchestrator uses this feedback to account for implementation-level performance and generate improved candidates in subsequent iterations.

### B. Construction of key components

The capabilities of OmniQEC are determined by its constituent agents and tools. Here, we present the key components implemented in the current framework, which are tailored to QEC discovery for *quantum-memory experiments* [17, 20, 21]. Refer to SI B for more details. As a flexible framework, its modular design allows additional agents and tools to be incorporated for other QEC tasks, such as the optimization of logical operations [65, 66], lattice-surgery protocols [67–69], and other fault-tolerant primitives [70–72].

*Orchestrator.* The orchestrator serves as the control layer of the OmniQEC framework. It translates a high-level design objective into a structured specification of the target code space, resource constraints and evaluation protocol, and routes tasks between the search agent and the scientific modules [51, 52, 73].

This functionality is realized using a general-purpose LLM backbone, such as GPT, Claude, or DeepSeek, equipped with task-specific prompts, tool-use interfaces, and access to the accumulated discovery history. For each candidate construction in each iteration, the orchestrator maintains a unified record containing its construction program, code-level properties, compiled circuits and decoding results. It distils this accumulated evidence into an evolving search context, enabling each iteration to generate better-informed constructions and progressively refine the discovery trajectory.

*Code-search agent.* This agent operates within the fast loop and proposes diverse executable QEC construction programs through LLM-guided program search [51, 52, 73]. We implement this agent using a dedicated LLM backbone,

such as GPT or DeepSeek. It receives the structured design objective and evolving search context from the orchestrator, and draws on QEC-specific knowledge of code constructions, algebraic constraints, and established design principles to formulate and revise *candidate programs*. Here each program refers to a construction rule together with its tunable parameters. For a CSS construction, for example, the rule may define how the binary check matrices $H_X$ and $H_Z$ are generated, while the tunable parameters control properties such as matrix size, polynomial support, sparsity pattern, and random seed [14, 17, 63]. A single program can therefore generate a family of related codes across different parameter choices. This programmatic representation preserves the provenance of each candidate and enables promising constructions to be systematically refined across iterations.

*Code-characterization module.* This module, as another key component in the fast loop, provides a general verification and evaluation interface between program generation and circuit-level analysis. It executes each construction program, checks whether the resulting code satisfies the defining algebraic conditions of the specified code family, and enforces the user-specified resource constraints. For CSS codes, for example, validity includes verifying the commutation condition $H_X H_Z^{\mathsf{T}} = 0$. Exact linear-algebra routines [3, 41] are then used to compute certified quantities such as the block length $n$, the number of logical qubits $k$, check weights, and qubit degrees. Quantities that are generally more difficult to determine exactly, most notably the code distance $d$, are estimated or bounded using in-house or open-source decoder-assisted tools, including BP+LSD [74] and OSD [50]. The modular interface also allows additional characterization tools to be incorporated as the framework evolves.

*Circuit-synthesis module.* This module operates within the slow loop and maps each selected code to an executable syndrome-extraction implementation required for circuit-level evaluation. Guided by the orchestrator, it allocates data qubits and ancillas, prepares the encoded state, and determines the protocol for repeated syndrome-extraction rounds. The protocol specifies how each stabilizer measurement is decomposed into native operations and how the resulting interactions are ordered and scheduled. For CSS codes, this entails separate measurements of the $X$- and $Z$-type checks specified by $H_X$ and $H_Z$.

The circuit-synthesis module in OmniQEC is implemented by integrating two complementary libraries. Within each syndrome-extraction round, the required two-qubit interactions are scheduled through edge colouring of the associated Tanner graph [43], producing conflict-free parallel circuit layers. The resulting schedule is then compiled into a complete Stim circuit [58] comprising state-preparation and reset operations, native entangling gates, ancilla measurements, detector definitions and logical-observable annotations. The module also returns circuit-level metrics, such as syndrome-extraction depth and two-qubit-gate count, to the orchestrator as feedback for subsequent discovery iterations.

*Circuit-evaluation module.* This module estimates the logical performance of each synthesized implementation under a configurable circuit-level noise model. The physical error rate can be specified by the user or chosen automatically at a common operating point slightly below the estimated pseudo-threshold, placing the evaluation in the error-suppression regime while retaining a sufficient number of logical failures for tractable direct sampling [64].

The implementation of this module consists of circuit sampling and modular decoding. For each synthesized implementation, the noisy Stim circuit [58] is converted into a detector error model, and sampled to generate detection events and logical-observable outcomes. These samples are then passed to a decoder-agnostic interface that can accommodate a broad range of decoding algorithms, including matching-based, belief-propagation-based, and neural decoders. In the current framework, OmniQEC integrates BP+OSD [50, 75], BP+LSD [74] and Relay-BP [76]. The resulting LER estimates and associated decoding records define the circuit-level ranking, which is returned to the orchestrator.

## III. EXPERIMENTAL RESULTS

We apply OmniQEC to the discovery of practical qLDPC codes under explicit hardware budgets. We focus on qLDPC codes because their finite-rate encoding and sparse parity checks offer a promising route to reducing the physical-qubit overhead of fault-tolerant quantum computation [16, 17, 77]. Practicality is enforced by restricting the total number of physical qubits $N$ to a scale comparable to the BB code with $[\![144, 12, 12]\!]$, which is relevant to early fault-tolerant quantum processors [17, 27]. The discovery of new qLDPC codes that match or surpass leading existing constructions at the circuit level, as measured by lower LER $p_{\mathrm{LER}}$, could broaden the range of fault-tolerant architectures realizable under realistic hardware constraints and accelerate progress towards practical fault-tolerant quantum computing [17, 78].

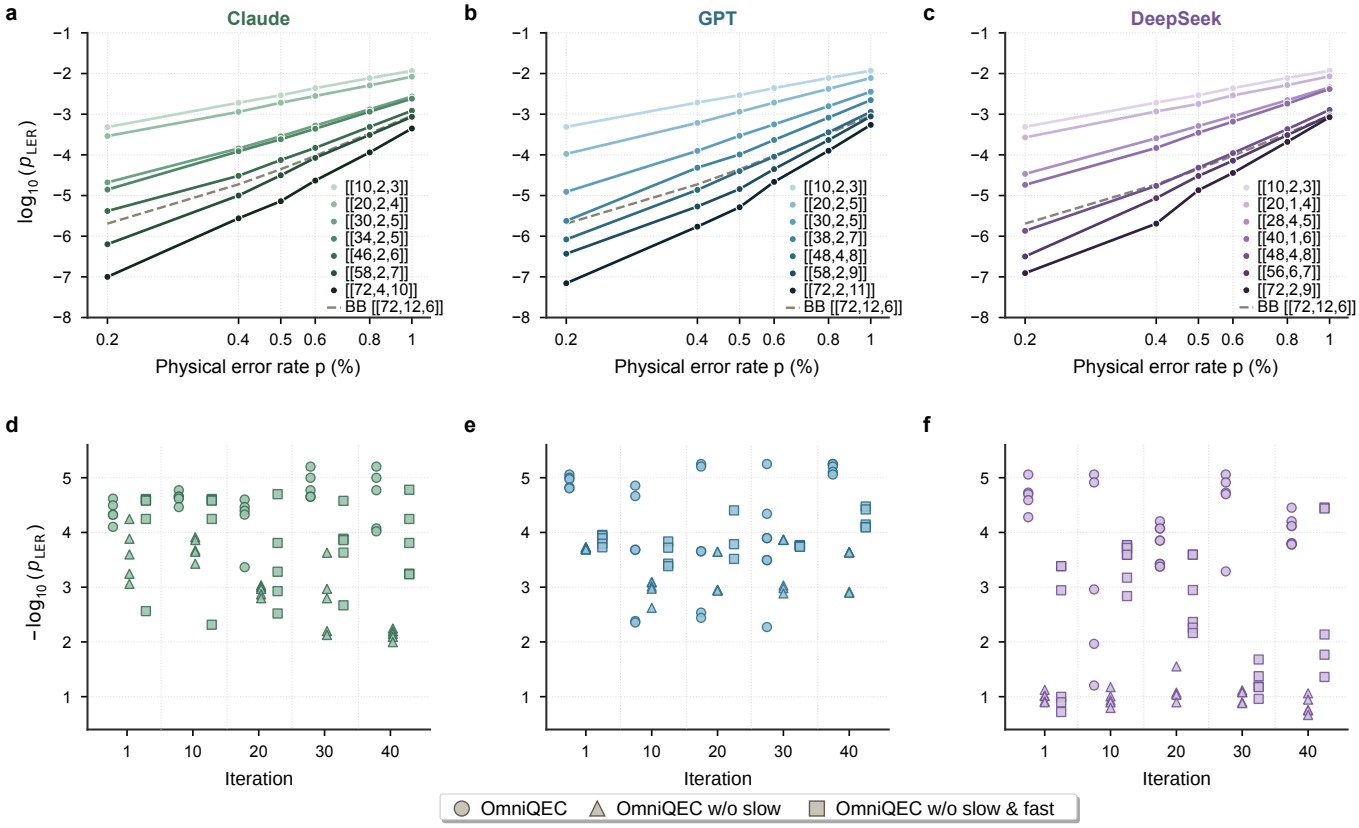


FIG. 3. **Circuit-level validation and loop ablation. a–c,** Physical-error-rate sweeps for representative frontier codes discovered by OmniQEC using Claude Opus 4.8 (**a**), GPT-5.5 (**b**) and DeepSeek-V4-Pro (**c**). The BB $[\![72, 12, 6]\!]$ code is shown as a reference under the same syndrome-extraction and decoding protocol. Solid lines denote OmniQEC-discovered codes and dashed lines denote the BB reference. The fitted scaling $p_{\mathrm{LER}} \propto p^{\Gamma}$ gives the LER-scaling exponent $\Gamma$. **d–f,** Ablation of discovery feedback for Claude Opus 4.8 (**d**), GPT-5.5 (**e**) and DeepSeek-V4-Pro (**f**). Each marker denotes one of the top-five candidates obtained at the corresponding iteration, evaluated by the same circuit-level protocol and plotted by its error suppression $-\log_{10}(p_{\mathrm{LER}})$. The full OmniQEC workflow uses both fast-loop code-level screening and slow-loop circuit-level feedback. The fast-loop-only variant, denoted by OmniQEC w/o slow, removes circuit-level feedback and ranks candidates using only code-level rankings. The no-feedback variant, denoted by OmniQEC w/o slow & fast, removes both feedback paths, so each iteration proceeds independently of previous results. The full workflow concentrates candidates in the high-suppression regime, whereas removing feedback increases variability and can steer the search towards circuit-level suboptimal candidates.

### A. Experiment setup and hyperparameter settings

We conduct systematic experiments to evaluate the capability of OmniQEC to discover practical qLDPC codes against the leading BB reference codes with $[\![72, 12, 6]\!]$ and $[\![144, 12, 12]\!]$, whose circuit implementations require total physical-qubit budgets of $N = 144$ and $N = 288$, respectively. Specifically, the orchestrator of OmniQEC is instantiated with three API-accessed LLM backends, which are Claude Opus 4.8, DeepSeek-V4-Pro, and GPT-5.5. The code-search agent explores four qLDPC construction families, i.e., BB, trivariate-bicycle (TB), lifted-product (LP), and generalized LP (GLP) codes, as detailed in SI A. We consider 14 total physical-qubit budgets spanning $N = 20$ to $N = 288$.

The hyperparameter settings employed in OmniQEC are fixed across all resource budgets and LLM backends. For each $N$, OmniQEC is executed for $T = 60$ dual-loop iterations. At each iteration, the code-search agent generates 24 construction programs, each instantiated with 20 random seeds, yielding up to 480 candidate code instances before validity checking and deduplication. The resulting code instances are assessed in the fast loop using the code-level objective $\Phi_{\mathrm{code}}$, after which a subset of the highest-ranking candidates is selected adaptively for slow-loop evaluation. In the slow loop, these candidates are compiled into syndrome-extraction memory circuits and evaluated using BP+OSD under circuit-level depolarizing noise with physical error rate $p = 0.005$. Unless stated otherwise, we report the LER per syndrome-extraction round and per logical qubit, i.e., $p_{\mathrm{LER}}$. Refer to SI C for a detailed definition for $p_{\mathrm{LER}}$ and other experimental details.

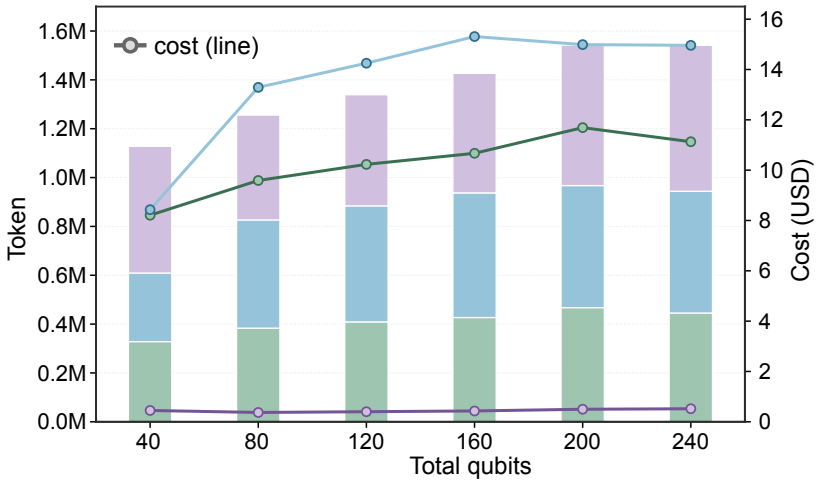


FIG. 4. **LLM-side token usage and API cost.** Bars show total API token consumption, including input, output and reasoning tokens, and lines show the corresponding API cost for each backend. Colors denote the LLM backend, using the same backend color coding as in Fig. 3.

### B. Scaling performance of OmniQEC

We first evaluate the performance of OmniQEC across different physical-qubit budgets $N$, with the results illustrated in Fig. 2. For all three LLM backends, the best discovered codes form a clear resource–performance frontier, with the LER $p_{\mathrm{LER}}$ decreasing approximately exponentially as $N$ increases. More specifically, all three backends surpass the BB $[\![72, 12, 6]\!]$ reference, with the first improvement occurring at $N = 98$. This crossover point is particularly relevant because it lies within the hardware scale of current reconfigurable processors, such as Helios [79]. However, at larger physical-qubit budgets, the performance frontiers diverge across backends. DeepSeek yields the weakest frontier and does not surpass the BB $[\![144, 12, 12]\!]$ reference within the explored range, whereas both GPT and Claude exceed this reference at $N = 240$. These differences highlight the role of the orchestrator backbone, as stronger reasoning and reflection capabilities enable circuit-level feedback to be converted more effectively into improved construction proposals.

A complementary analysis of the discovered code families, presented in SI D, reveals distinct exploration behaviours across orchestrators. The leading candidates identified by Claude and GPT are dominated by BB constructions, whereas DeepSeek retains greater diversity across multiple code families but attains a weaker performance frontier. This contrast highlights an exploration–exploitation trade-off in OmniQEC and suggests a direction for future improvement. That is, broad exploration preserves structural diversity, whereas effective exploitation depends on the orchestrator recognizing when circuit-level evidence is sufficiently consistent to focus subsequent search on high-performing code structures.

Another finding from the scaling analysis is that the circuit-level performance frontier is poorly aligned with the fast-loop objective. The proxy $\Phi_{\mathrm{code}}$, which is widely adopted in previous code-centric code searches [36–38], assigns scores of 6 and 12 to the BB $[\![72, 12, 6]\!]$ and $[\![144, 12, 12]\!]$ references, respectively. Yet only one code discovered by OmniQEC exceeds the larger reference under this metric. Despite their lower proxy scores, many codes discovered at $N \geq 98$ achieve lower circuit-level LER than the BB $[\![72, 12, 6]\!]$ reference. The mismatch is especially evident at $N = 144$, where the best codes found with DeepSeek, Claude and GPT have proxy scores of $\Phi_{\mathrm{code}} = 2.25$, 5.56 and 3.36, respectively, while all outperform the BB reference in circuit-level LER. This rank mismatch underlines the need to use LER as a direct objective in practical QEC code design, beyond relying on code-level proxies alone.

### C. Circuit-level validation of OmniQEC

We next test whether the advantage of the discovered codes persists across physical error rates $p$. To this end, we select seven frontier codes discovered by OmniQEC for each LLM backend and perform full physical-error-rate sweeps, yielding 21 codes in total. We re-evaluate these codes and the BB $[\![72, 12, 6]\!]$ reference over $p \in [0.002, 0.01]$ using identical syndrome-extraction and decoding protocols. As shown in Fig. 3a–c, the discovered codes maintain lower LER than the BB reference throughout the measured range. At the two endpoints, the strongest candidate achieves

an LER that is 29.23-fold lower at $p = 0.002$ and 1.66-fold lower at $p = 0.01$.

We further quantify the dependence of the LER $p_{\mathrm{LER}}$ on $p$ by fitting $p_{\mathrm{LER}} \propto p^{\Gamma}$, where $\Gamma$ is the fitted LER-scaling exponent. The representative codes, i.e., $[\![48, 4, 8]\!]$ discovered with GPT, $[\![58, 2, 7]\!]$ discovered with Claude and $[\![48, 4, 8]\!]$ discovered with DeepSeek, yield $\Gamma = 5.20$, 5.60 and 5.57, respectively. Compared with $\Gamma = 3.79$ for the BB reference, these results show that the advantage of the OmniQEC-discovered codes persists beyond the single physical error rate used during discovery.

### D. Ablation study of OmniQEC

We then move on to comprehend the respective roles of the fast and slow loops through ablation experiments. To be concrete, we compare the full OmniQEC workflow with two reduced variants. The fast-loop-only variant, denoted by OmniQEC w/o slow, removes the circuit-level feedback, so the discovery process is solely based on code-level rankings. The no-feedback variant, denoted by OmniQEC w/o slow & fast, removes both feedback paths, causing each iteration to proceed independently of previous results. For a fair comparison, the top five candidates from each variant are evaluated using the same circuit-level protocol.

The achieved results are demonstrated in Fig. 3d–f. The full OmniQEC workflow consistently attains stronger and more stable error suppression than its ablated variants. At the final iteration, the top-five candidates reach mean error suppressions of $-\log_{10}(p_{\mathrm{LER}}) = 4.61 \pm 0.54$, $5.17 \pm 0.09$, and $4.07 \pm 0.28$ for Claude Opus 4.8, GPT-5.5, and DeepSeek-V4-Pro, respectively. Compared with the full workflow, OmniQEC w/o slow reaches only $2.14 \pm 0.10$, $3.20 \pm 0.40$, and $0.84 \pm 0.16$, while OmniQEC w/o slow & fast reaches $3.86 \pm 0.66$, $4.25 \pm 0.19$, and $2.83 \pm 1.50$. Across the three backends and all checkpoints, the full workflow further achieves the highest mean suppression, 4.27, and the lowest variance, 0.79. Removing feedback increases the variance substantially to 1.24–1.26. Taken together, these results show that the fast loop improves search efficiency, whereas the slow loop is essential for steering discovery towards practically relevant circuit-level performance.

### E. Implementation cost of OmniQEC

We lastly assess the LLM-side cost of OmniQEC across different backends. As shown in Fig. 4, API token usage varies only weakly with the physical-qubit budget $N$, indicating that larger code instances do not substantially increase the required context. When considered alongside the achieved performance frontiers, Claude offers the most favourable cost–performance balance, whereas DeepSeek is the least expensive but produces the weakest frontier. These results indicate that LLM-side cost is governed primarily by backend-specific reasoning behaviour and API pricing, rather than by the size of the codes being explored.

## IV. DISCUSSION

In this study, we introduce OmniQEC, a new paradigm for automated QEC code discovery. Unlike prior approaches that optimize code-level proxies, OmniQEC directly targets implementation-level performance. To make this objective computationally tractable, OmniQEC harnesses a purpose-built self-evolving reasoning mechanism, specialized scientific tools and a slow–fast synergistic workflow. OmniQEC thereby provides a scalable route towards discovering practical, high-performing QEC codes.

Our study points to several broader directions for autonomous QEC design. A first direction is to develop more capable and efficient scientific tools for OmniQEC. Its main computational bottleneck is circuit-level verification, which limits the number of candidates that can be evaluated using physical evidence. One potential route is to improve key components such as syndrome-extraction synthesis [43, 44], circuit simulation and decoding [58, 80–82], together with more efficient statistical estimation of logical performance. Because these components are exposed through modular interfaces, new tools could be integrated directly into OmniQEC, enabling broader searches over richer and more hardware-relevant objectives [80–82].

Moreover, it is important to extend OmniQEC from its present agent–module architecture towards an end-to-end multi-agent system [83–85]. At present, only the orchestrator and code-search agent use LLM backends, whereas code characterization, circuit synthesis, and circuit evaluation invoke predefined tools. As the toolset grows, these modules could be upgraded into specialized agents that adapt circuit schedules, flag-qubit placement and decoder configurations to each candidate code. This would enable increasingly autonomous co-design of codes, circuits and decoders, while keeping evaluation grounded in executable tools and verifiable evidence [86–88].

Another important direction is to extend OmniQEC towards broader hardware objectives and code families. Recent work has shown that QEC detection events can provide learning signals for continuously adjusting hardware-control parameters [89]. In OmniQEC, calibration data such as gate-error maps and disabled qubits or couplers could instead guide code selection, physical embedding, syndrome-extraction scheduling, and decoder configuration. This would extend the present hardware-aware search towards calibration-aware code–circuit–decoder co-design for a specific processor. The search space could also expand beyond the four families studied here to higher-dimensional hypergraph-product codes [90], two-block group-algebra codes [91], balanced-product codes [15] and Floquet codes [9, 10]. Together, richer objectives and construction templates would enable OmniQEC to discover codes tailored to neutral-atom arrays, trapped-ion processors, and other reconfigurable platforms [22–25].

A parallel direction is to advance the entire OmniQEC pipeline towards a hierarchical multi-agent system, in which specialized agents coordinate under a central scientific orchestrator [83–85]. For example, a circuit-synthesis agent could compare candidate schedules and optimize circuit depth or ancilla overhead, whereas a decoding agent could select decoder families and tune their configurations for each code. An independent evaluation agent could further assess intermediate proposals, although the known biases of LLM-as-a-judge approaches motivate grounding such assessments in executable tools and verifiable physical evidence [86–88]. Such an architecture could extend OmniQEC from tool-assisted code discovery to increasingly autonomous code–circuit–decoder co-design.


[1] D. Gottesman, An introduction to quantum error correction and fault-tolerant quantum computation, arXiv preprint arXiv:0904.2557 (2009).
[2] B. M. Terhal, Quantum error correction for quantum memories, Reviews of Modern Physics **87**, 307 (2015).
[3] J. Roffe, Quantum error correction: an introductory guide, Contemporary Physics **60**, 226 (2019).
[4] A. Y. Kitaev, Fault-tolerant quantum computation by anyons, Annals of Physics **303**, 2 (2003).
[5] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, Journal of Mathematical Physics **43**, 4452 (2002).
[6] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, Physical Review A—Atomic, Molecular, and Optical Physics **86**, 032324 (2012).
[7] H. Bombin and M. A. Martin-Delgado, Topological quantum distillation, Physical review letters **97**, 180501 (2006).
[8] N. Lacroix, A. Bourassa, F. J. Heras, L. M. Zhang, J. Bausch, A. W. Senior, T. Edlich, N. Shutty, V. Sivak, A. Bengtsson, *et al.*, Scaling and logic in the color code on a superconducting quantum processor, Nature , 1 (2025).
[9] M. B. Hastings and J. Haah, Dynamically generated logical qubits, Quantum **5**, 564 (2021).
[10] M. Davydova, N. Tantivasadakarn, and S. Balasubramanian, Floquet codes without parent subsystem codes, PRX Quantum **4**, 020341 (2023).
[11] C. Gidney, M. Newman, A. Fowler, and M. Broughton, A fault-tolerant honeycomb memory, Quantum **5**, 605 (2021).
[12] H. Cao, G. Yan, Y. Du, and F. Pan, Maximum likelihood decoding of quantum error correction codes, arXiv preprint arXiv:2605.17230 (2026).
[13] J.-P. Tillich and G. Zémor, Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength, IEEE Transactions on Information Theory **60**, 1193 (2014).
[14] P. Panteleev and G. Kalachev, Quantum ldpc codes with almost linear minimum distance, IEEE Transactions on Information Theory **68**, 213 (2021).
[15] N. P. Breuckmann and J. N. Eberhardt, Balanced product quantum codes, IEEE Transactions on Information Theory **67**, 6653 (2021).
[16] P. Panteleev and G. Kalachev, Asymptotically good quantum and locally testable classical ldpc codes, in *Proceedings of the 54th annual ACM SIGACT symposium on theory of computing* (2022) pp. 375–388.
[17] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, High-threshold and low-overhead fault-tolerant quantum memory, Nature **627**, 778 (2024).
[18] H. Goto, High-performance fault-tolerant quantum computing with many-hypercube codes, Science Advances **10**, eadp6388 (2024).
[19] D. Ruiz, J. Guillaud, A. Leverrier, M. Mirrahimi, and C. Vuillot, Ldpc-cat codes for low-overhead quantum computing in 2d, Nature Communications **16**, 1040 (2025).
[20] G. Q. AI, Suppressing quantum errors by scaling a surface code logical qubit, Nature **614**, 676 (2023).
[21] G. Q. AI and Collaborators, Quantum error correction below the surface code threshold, Nature **638**, 920 (2025).
[22] D. Bluvstein *et al.*, Logical quantum processor based on reconfigurable atom arrays, Nature **626**, 58 (2024).
[23] Q. Xu, J. P. Bonilla Ataides, C. A. Pattison, N. Raveendran, D. Bluvstein, J. Wurtz, B. Vasić, M. D. Lukin, L. Jiang, and H. Zhou, Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays, Nature Physics **20**, 1084 (2024).
[24] L. Egan, D. M. Debroy, C. Noel, A. Risinger, D. Zhu, D. Biswas, M. Newman, M. Li, K. R. Brown, M. Cetina, and C. Monroe, Fault-tolerant control of an error-corrected qubit, Nature **598**, 281 (2021).
[25] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, *et al.*, Realization of real-time fault-tolerant quantum error correction, Physical Review X **11**, 041058 (2021).

[26] S. Bravyi, D. Poulin, and B. Terhal, Tradeoffs for reliable quantum information storage in 2d systems, Physical review letters **104**, 050503 (2010).

[27] A. Katabarwa, K. Gratsea, A. Caesura, and P. D. Johnson, Early fault-tolerant quantum computing, PRX quantum **5**, 020101 (2024).

[28] H. Zhou, M. Cain, and M. D. Lukin, Opportunities in full-stack design of low-overhead fault-tolerant quantum computation, Nature Computational Science **5**, 1110 (2025).

[29] H. P. Nautrup, N. Delfosse, V. Dunjko, H. J. Briegel, and N. Friis, Optimizing quantum error correction codes with reinforcement learning, Quantum **3**, 215 (2019).

[30] J. Olle, R. Zen, M. Puviani, and F. Marquardt, Simultaneous discovery of quantum error correction codes and encoders with a noise-aware reinforcement learning agent, npj Quantum Information **10**, 126 (2024).

[31] V. P. Su, C. Cao, H.-Y. Hu, Y. Yanay, C. Tahan, and B. Swingle, Discovery of optimal quantum codes via reinforcement learning, Physical review applied **23**, 034048 (2025).

[32] B. C. A. Freire, N. Delfosse, and A. Leverrier, Optimizing hypergraph product codes with random walks, simulated annealing and reinforcement learning, in *2025 IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2025) pp. 1–6.

[33] A. Y. He and Z.-W. Liu, Discovering highly efficient low-weight quantum error-correcting codes with reinforcement learning, arXiv preprint arXiv:2502.14372 (2025).

[34] Y. Du, Y. Zhu, Y.-H. Zhang, M.-H. Hsieh, P. Rebentrost, W. Gao, Y.-D. Wu, J. Eisert, G. Chiribella, D. Tao, *et al.*, Artificial intelligence for representing and characterizing quantum systems, arXiv preprint arXiv:2509.04923 (2025).

[35] Y. Chengyu, R. Meister, C. Carty, S.-K. Lin, and R. Bondesan, Bayesian optimization for quantum error-correcting code discovery, arXiv preprint arXiv:2601.18562 (2026).

[36] J. Cruz-Benito, A. W. Cross, D. Kremer, and I. Faro, Evolutionary discovery of bivariate bicycle codes with llm-guided search, arXiv preprint arXiv:2606.02418 (2026).

[37] X. He, S. Lu, and B. Zeng, Co-designing quantum codes with transversal diagonal gates via multi-agent systems, in *2026 IEEE 19th Dallas Circuits and Systems Conference (DCAS)* (IEEE, 2026) pp. 1–6.

[38] Z. Liu and F. Marquardt, Large-language-model discovery of quantum ldpc codes through structured concept evolution, arXiv preprint arXiv:2606.24808 (2026).

[39] A. R. Calderbank and P. W. Shor, Good quantum error-correcting codes exist, Physical Review A **54**, 1098 (1996).

[40] A. Steane, Multiple-particle interference and quantum error correction, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **452**, 2551 (1996).

[41] D. Gottesman, *Stabilizer codes and quantum error correction* (California Institute of Technology, 1997).

[42] O. Higgott, T. C. Bohdanowicz, A. Kubica, S. T. Flammia, and E. T. Campbell, Improved decoding of circuit noise and fragile boundaries of tailored surface codes, Physical Review X **13**, 031007 (2023).

[43] K. Zhang, D. Gao, Z. Yang, R. Zhou, F. Liu, Z. Ji, and J. Chen, Optimal compilation of syndrome extraction circuits for general quantum ldpc codes, in *2026 Design, Automation & Test in Europe Conference (DATE)* (IEEE, 2026) pp. 1–7.

[44] J. Viszlai, S. Maurya, S. Tannu, M. Martonosi, and F. T. Chong, Prophunt: Automated optimization of quantum syndrome measurement circuits, in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (2026) pp. 1476–1491.

[45] Z. Liang, K. Liu, H. Song, and Y.-A. Chen, Generalized toric codes on twisted tori for quantum error correction, PRX Quantum **6**, 020357 (2025).

[46] N. Baspin and A. Krishna, Connectivity constrains quantum codes, Quantum **6**, 711 (2022).

[47] A. Strikis, D. E. Browne, and M. E. Beverland, High-performance syndrome extraction circuits for quantum codes, arXiv preprint arXiv:2603.05481 (2026).

[48] G. Kishony and A. Fowler, Surface code off-the-hook: diagonal syndrome-extraction scheduling, arXiv preprint arXiv:2602.09099 (2026).

[49] R. Bi, K. Chang, and S. Puri, Untangling qldpc codes with biased noise ancilla, arXiv preprint arXiv:2606.30592 (2026).

[50] J. Roffe, D. R. White, S. Burton, and E. Campbell, Decoding across the quantum low-density parity-check code landscape, Physical Review Research **2**, 043423 (2020).

[51] D. A. Boiko, R. MacKnight, B. Kline, and G. Gomes, Autonomous chemical research with large language models, Nature **624**, 570 (2023).

[52] C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, and D. Ha, The ai scientist: Towards fully automated open-ended scientific discovery, arXiv preprint arXiv:2408.06292 (2024).

[53] Z. Chen, A. N. Petsch, A. J. Israelski, R. Plumley, L. Shen, C. Wang, C. Peng, Y. Ni, A. Bansil, S. Chowdhury, *et al.*, An agentic artificially intelligent x-ray scientist, Nature Machine Intelligence , 1 (2026).

[54] A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller, Augmenting large language models with chemistry tools, Nature machine intelligence **6**, 525 (2024).

[55] S. Arlt, H. Duan, F. Li, S. M. Xie, Y. Wu, and M. Krenn, Meta-designing quantum experiments with language models, Nature Machine Intelligence **8**, 148 (2026).

[56] Y. Kim, K. Gu, C. Park, C. Park, S. Schmidgall, A. A. Heydari, Y. Yan, Z. Zhang, Y. Zhuang, Y. Liu, M. Malhotra, P. P. Liang, H. W. Park, Y. Yang, X. Xu, Y. Du, S. Patel, T. Althoff, D. McDuff, and X. Liu, Capable language models can outgrow the benefits of collaboration, Nature Machine Intelligence **8**, 1157 (2026).

[57] A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. Ruiz, A. Mehrabian, *et al.*, Alphaevolve: A coding agent for scientific and algorithmic discovery, arXiv preprint arXiv:2506.13131 (2025).

[58] C. Gidney, Stim: a fast stabilizer circuit simulator, Quantum **5**, 497 (2021).
[59] S. Yao, J. Zhao, D. Yu, I. Shafran, K. R. Narasimhan, and Y. Cao, React: Synergizing reasoning and acting in language models, in *NeurIPS 2022 Foundation Models for Decision Making Workshop* (2022).
[60] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, Reflexion: Language agents with verbal reinforcement learning, Advances in neural information processing systems **36**, 8634 (2023).
[61] G. Yan, W. Wu, Y. Chen, K. Pan, X. Lu, Z. Zhou, Y. Wang, R. Wang, and J. Yan, Ai-empowered quantum circuit synthesis and compilation optimization: Overview and prospects, IEEE Transactions on Pattern Analysis and Machine Intelligence (2026).
[62] P. Panteleev and G. Kalachev, Asymptotically good quantum and locally testable classical ldpc codes, arXiv preprint arXiv:2111.03654 (2021).
[63] L. Voss, S. J. Xian, T. Haug, and K. Bharti, Multivariate bicycle codes, Physical Review A **111**, L060401 (2025).
[64] C. Mayer, A. Ganti, U. Onunkwo, T. Metodi, B. Anker, and J. Skryzalin, Rare event simulation of quantum error-correcting circuits, arXiv preprint arXiv:2509.13678 (2025).
[65] G. Zhang and Y. Li, Time-efficient logical operations on quantum low-density parity check codes, Physical Review Letters **134**, 070602 (2025).
[66] M. Cain, C. Zhao, H. Zhou, N. Meister, J. Ataides, A. Jaffe, D. Bluvstein, and M. D. Lukin, Correlated decoding of logical algorithms with transversal gates, arXiv preprint arXiv:2403.03272 (2024).
[67] C. Horsman, A. G. Fowler, S. Devitt, R. Van Meter, *et al.*, Surface code quantum computing by lattice surgery, New Journal of physics **14**, 123011 (2012).
[68] D. Litinski, A game of surface codes: Large-scale quantum computing with lattice surgery, Quantum **3**, 128 (2019).
[69] C. Chamberland and E. T. Campbell, Universal quantum computing with twist-free and temporally encoded lattice surgery, PRX Quantum **3**, 010331 (2022).
[70] S. Bravyi and A. Kitaev, Universal quantum computation with ideal clifford gates and noisy ancillas, Physical Review A—Atomic, Molecular, and Optical Physics **71**, 022316 (2005).
[71] H. Bombín, Single-shot fault-tolerant quantum error correction, Physical Review X **5**, 031043 (2015).
[72] C. Gidney and A. G. Fowler, Efficient magic state factories with a catalyzed $|ccz\rangle$ to $2|t\rangle$ transformation, Quantum **3**, 135 (2019).
[73] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, *et al.*, Mathematical discoveries from program search with large language models, Nature **625**, 468 (2024).
[74] T. Hillmann, L. Berent, A. O. Quintavalle, J. Eisert, R. Wille, and J. Roffe, Localized statistics decoding for quantum low-density parity-check codes, Nature Communications **16**, 8214 (2025).
[75] P. Panteleev and G. Kalachev, Degenerate quantum ldpc codes with good finite length performance, Quantum **5**, 585 (2021).
[76] T. Müller, T. Alexander, M. E. Beverland, M. Bühler, B. R. Johnson, T. Maurer, and D. Vandeth, Improved belief propagation is sufficient for real-time decoding of quantum memory, arXiv preprint arXiv:2506.01779 (2025).
[77] N. P. Breuckmann and J. N. Eberhardt, Quantum low-density parity-check codes, PRX quantum **2**, 040101 (2021).
[78] E. T. Campbell, B. M. Terhal, and C. Vuillot, Roads towards fault-tolerant universal quantum computation, Nature **549**, 172 (2017).
[79] A. Ransford, M. Allman, J. Arkinstall, J. Campora, S. F. Cooper, R. D. Delaney, J. M. Dreiling, B. Estey, C. Figgatt, A. Hall, *et al.*, A 98-qubit trapped-ion quantum computer with all-to-all connectivity, Nature , 1 (2026).
[80] A. W. Senior, T. Edlich, F. J. Heras, L. M. Zhang, O. Higgott, J. S. Spencer, T. Applebaum, S. Blackwell, J. Ledford, A. Žemgulytė, *et al.*, A scalable and real-time neural decoder for topological quantum codes, arXiv preprint arXiv:2512.07737 (2025).
[81] G. Yan, S. Li, and Y. Du, Rethink the role of neural decoders in quantum error correction, arXiv preprint arXiv:2605.12046 (2026).
[82] G. Yan, S. Li, S. Xiao, P. Ma, H. Cao, F. Pan, and Y. Du, Efficient foundation decoders for fault-tolerant quantum computing, arXiv preprint arXiv:2606.27119 (2026).
[83] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, Camel: Communicative agents for" mind" exploration of large language model society, Advances in neural information processing systems **36**, 51991 (2023).
[84] G. Zhang, L. Niu, J. Fang, K. Wang, L. Bai, and X. Wang, Multi-agent architecture search via agentic supernet, in *International Conference on Machine Learning* (PMLR, 2025) pp. 75834–75852.
[85] G. Zhang, H. Geng, X. Yu, Z. Yin, Z. Zhang, Z. Tan, H. Zhou, Z. Li, X. Xue, Y. Li, *et al.*, The landscape of agentic reinforcement learning for llms: A survey, arXiv preprint arXiv:2509.02547 (2025).
[86] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, *et al.*, Judging llm-as-a-judge with mt-bench and chatbot arena, Advances in neural information processing systems **36**, 46595 (2023).
[87] J. D'Souza, H. Babaei Giglou, and Q. Münch, YESciEval: Robust LLM-as-a-judge for scientific question answering, in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, edited by W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar (Association for Computational Linguistics, Vienna, Austria, 2025) pp. 13749–13783.
[88] A. Findeis, F. Weers, G. Yin, K. Ye, R. Pang, and T. Gunter, Can external validation tools improve annotation quality for LLM-as-a-judge?, in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, edited by W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar (Association for Computational Linguistics, Vienna, Austria, 2025) pp. 15997–16020.
[89] V. Sivak, A. Morvan, M. Broughton, R. G. Cortiñas, J. Bausch, A. W. Senior, M. Neeley, A. Eickbusch, N. Shutty, L. A.

Beni, *et al.*, Reinforcement learning control of quantum error correction, Nature , 1 (2026).
[90] W. Zeng and L. P. Pryadko, Higher-dimensional quantum hypergraph-product codes with finite rates, Physical review letters **122**, 230501 (2019).
[91] H.-K. Lin and L. P. Pryadko, Quantum two-block group algebra codes, Physical Review A **109**, 022407 (2024).