MedDCR: Learning to Design Agentic Workflows for Medical Coding

Jiyang Zheng[1,2], Islam Nassar[1], Thanh Vu[1], Xu Zhong[1],
Yang Lin[1], Tongliang Liu[2], Long Duong[1], Yuan-Fang Li[1]
[1]Oracle Health and AI
[2]Sydney AI Center, The University of Sydney
{jiyang.zheng, islam.nassar, thanh.v.vu, peter.zhong,
yang.y.lin, long.duong, yuanfang.li}@oracle.com
{tongliang.liu}@sydney.edu.au

Abstract

Medical coding converts free-text clinical notes into standardized diagnostic and procedural codes, which are essential for billing, hospital operations, and medical research. Unlike ordinary text classification, it requires multi-step reasoning: extracting diagnostic concepts, applying guideline constraints, mapping to hierarchical codebooks, and ensuring cross-document consistency. Recent advances leverage agentic LLMs, but most rely on rigid, manually crafted workflows that fail to capture the nuance and variability of real-world documentation, leaving open the question of how to systematically learn effective workflows. We present ***MedDCR***, a closed-loop framework that treats workflow design as a learning problem. A Designer proposes workflows, a Coder executes them, and a Reflector evaluates predictions and provides constructive feedback, while a memory archive preserves prior designs for reuse and iterative refinement. On benchmark datasets, MedDCR outperforms state-of-the-art baselines and produces interpretable, adaptable workflows that better reflect real coding practice, improving both the reliability and trustworthiness of automated systems.

1 Introduction

Medical coding is the process of translating unstructured clinical notes into standardized diagnostic and procedural codes, most commonly following the World Health Organization's International Classification of Diseases (ICD) standard (Dong et al., 2022). These codes underpin critical functions in healthcare, including billing and reimbursement, hospital resource planning, and epidemiological research (Campbell and Giadresco, 2020). Unlike simple classification, manual coding requires coders to engage in a multi-step workflow: identifying relevant mentions in free text, consulting multiple resources such as the alphabetic index and tabular index, applying coding guidelines, and cross-checking for consistency across diagnoses and procedures. This structured but intricate process makes medical coding highly labour-intensive and error-prone, contributing to global coding backlogs and clinical risks when errors occur (Alonso et al., 2020; Douglas et al., 2025; Gan et al., 2025).

To reduce the burden of manual coding, recent research has leveraged large language models (LLMs) as the foundation for automated systems (Boyle et al., 2023; Yang et al., 2023b; Falis et al., 2024; Baksi et al., 2025). Beyond their core ability to map free-text clinical notes to candidate codes, LLMs possess reasoning and tool-use capabilities that are particularly well-suited to the structured, rule-based nature of the coding process (Kwan, 2024; Mustafa et al., 2025).

Building on this, emerging work has proposed agentic coding workflows (Li et al., 2024b; Motzfeldt et al., 2025), where multiple interacting agents cooperate to mirror the steps taken by human coders: extracting relevant terms, consulting indexes and guidelines, and verifying consistency across diagnoses and procedures. This paradigm has demonstrated competitive performance and enhanced interpretability compared to treating coding as a flat multi-label classification problem.

Despite these advances, most existing agentic frameworks for medical coding remain manually crafted, relying on human experts to specify the modules and execution steps within a workflow (Motzfeldt et al., 2025). Yet this *design problem* is particularly challenging, as correct code assignment with LLM-based systems often hinges on two factors: (i) the quality of module definitions (e.g., tool calls, inference strategies), and (ii) the effectiveness of interactions across modules (i.e., how tools and strategies are combined and ordered). Manually fixing these design choices risks overlooking more effective coordination patterns, thereby limiting the potential of agentic approaches (Li et al., 2024a; Zhang et al., 2025b).

Findings of the Association for Computational Linguistics: ACL 2026, pages 12878–12893
July 2-7, 2026

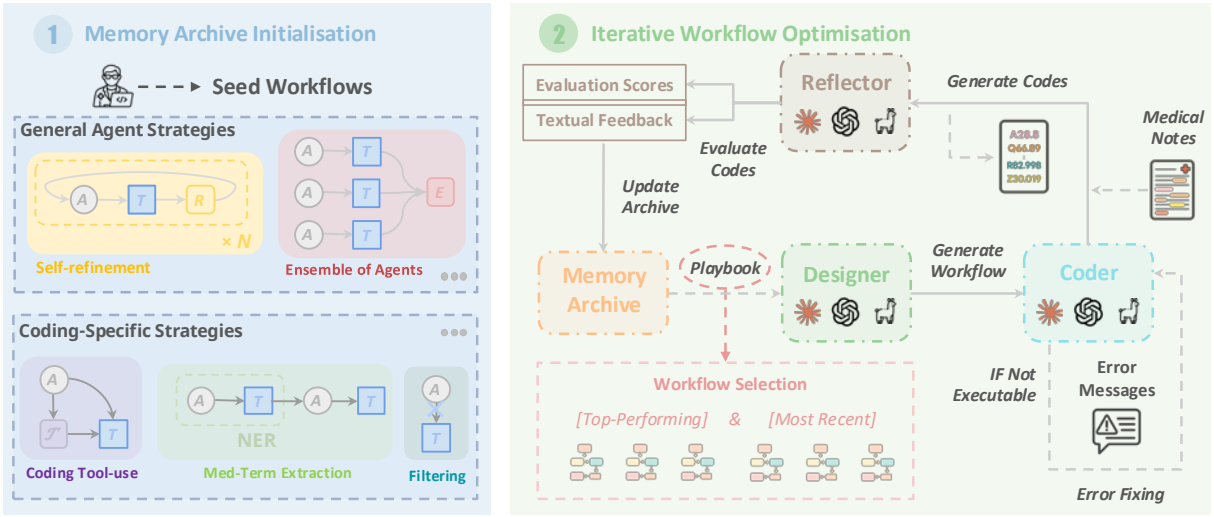

Figure 1: Overview of the **MedDCR** framework. (1) The *memory archive* is initialised with general reasoning strategies (e.g., self-refinement, multi-agent ensembles, chain-of-thought prompting) and coding-specific strategies (e.g., medical term extraction, weak code filtering, ICD tool use), together with other optional seed workflows. (2) In each optimisation loop, the *Designer* proposes new workflows, the *Coder* compiles and executes them (with self-fixing if needed), and the *Reflector* provides both evaluation scores and textual feedback. The memory archive stores all past workflows, enabling reuse, progressive refinement, and workflow selection from top-performing and recent designs. This closed-loop process discovers effective coding workflows under guideline constraints.

For instance, automated search may discover non-obvious yet beneficial strategies, such as applying a *contrastive screening* step to prune near-duplicate ICD codes based on description similarity, whereas manual designs may fail to recognize such refinements. This motivates the need for automated workflow learning, which can flexibly search for and refine workflow designs rather than constraining systems to static, expert-defined pipelines.

To address this challenge, we propose MedDCR (As demonstrated in Figure 1), an automated framework that optimises workflows for medical coding. Instead of relying on a single fixed pipeline, MedDCR treats workflow design as a learning problem (Hu et al., 2025; Zhang et al., 2025a,b; Zhou et al., 2025), where workflows are proposed, executed, and evaluated in an iterative loop.

Within this loop, a **D**esigner agent generates workflow plans by inventing or combining coding tools and strategies. A **C**oder agent then translates these plans into executable pipelines in the form of concrete programs, which carry out operations such as tool calling, validation, and reconciliation to predict medical codes. Finally, a **R**eflector agent evaluates the predictions, providing both performance scores and textual feedback on the effectiveness of the workflow design. The Designer uses this feedback to refine its proposals, enabling workflows to evolve and improve over time (Wang et al., 2025).

Beyond this loop, MedDCR maintains a memory archive of prior designs, enabling workflows to be reused, progressively refined, or augmented with expert-crafted pipelines (Ji et al., 2024; Li et al., 2024b; Motzfeldt et al., 2025).

Our experiments demonstrate that the workflows discovered by MedDCR significantly outperform both state-of-the-art, hand-designed baselines and pretrained language model approaches. Specifically, MedDCR achieves a 6.2% improvement in F1 score on the MDACE (Cheng et al., 2023) dataset and a 7.4% gain on ACI-BENCH (Yim et al., 2023), compared to the second-best performing baselines.

Building upon these insights and findings, we present our main contributions as follows:

- We propose MedDCR, a novel framework for medical coding that treats workflow design as a learning problem through iterative design–coding/execute–reflect loops.
- We develop a meta-agent architecture with Designer, Coder, and Reflector agents, supported by a memory archive that enables both search from scratch and plug-and-play optimisation of expert medical coding workflows.
- MedDCR demonstrates state-of-the-art performance on multiple benchmarks, along with improved interpretability and risk analysis.

2 Related Works

2.1 Automated Medical Coding

Automated medical coding seeks to accelerate the mapping of free-text clinical notes to standardized medical codes using computer-assisted tools. Early systems were rule-based (Farkas and Szarvas, 2008; Kavuluru et al., 2015), but the availability of large EHR datasets such as MIMIC-III and MIMIC-IV (Johnson et al., 2016, 2020) established deep learning models as the dominant approach, typically framing coding as an extreme multi-label classification task. Encoder–decoder architectures with label-wise attention (Mullenbach et al., 2018; Li and Yu, 2020; Vu et al., 2021) were extended with textual descriptions, synonyms, or co-occurrence signals to better align notes and codes (Cao et al., 2020; Dong et al., 2021; Yuan et al., 2022). More recently, pre-trained language models fine-tuned for ICD prediction (PLM-ICD) achieved state-of-the-art performance (Huang et al., 2022; Edin et al., 2024; Douglas et al., 2025). However, these models struggle with the extremely large ICD label space, rare codes, and long clinical notes.

To address these challenges, researchers have explored LLMs for generative coding. Zero and few-shot prompting (Yang et al., 2023c; Boyle et al., 2023; Gero et al., 2023) showed flexibility but underperformed PLM-based classifiers. This has motivated agentic approaches, where LLMs interact with external tools, indexes, and validation routines in multi-agent workflows that mimic human coding processes (Li et al., 2024b; Kwan, 2024; Motzfeldt et al., 2025). While promising, these workflows remain manually designed and fixed, making them potentially suboptimal.

Our work addresses this gap by introducing MedDCR, which treats workflow design as a learning problem and automatically searches for effective agentic workflows for medical coding, enhancing both predictive performance and interpretability.

2.2 Agentic Workflow Design

Agentic workflows frame problem solving as a coordinated process across one or more LLM-based agents, each assigned specific roles or equipped with external tools. Recent advances enrich these systems with prompting strategies (Chen et al., 2023), chain-of-thought planning (Wei et al., 2022; Zhang et al., 2023), reflection and refinement (Madaan et al., 2023; Shinn et al., 2023; Dhuliawala et al., 2024), tool use (Nakano et al., 2021; Schick et al., 2023; Qin et al., 2024), and role assignment for multi-agent cooperation (Shanahan et al., 2023; Li et al., 2023; Wu et al., 2024). Multi-agent topologies vary from parallel setups for exploration (Wang et al., 2023a) to serial pipelines with reflective refinement (Madaan et al., 2023), and even debate-style structures that improve reliability (Du et al., 2023; Liang et al., 2024).

Building on these foundations, the community has begun exploring automated design of agentic systems. Most works focus on automating prompt optimization (Yang et al., 2023a; Fernando et al., 2024; Khattab et al., 2024), role definition (Li et al., 2023; Wu et al., 2024), or specific topology search (Zhou et al., 2025; Wang et al., 2025). Others attempt to expand the search space to workflows (Hu et al., 2025; Zhang et al., 2025a,b), where both the definition of workflow components and their topological organisation are jointly optimised, but in practice, many components remain fixed, making discovered agents less flexible.

Our work advances this line by introducing MedDCR, which treats agentic workflow design as a learning problem. Unlike prior approaches, MedDCR employs a meta-agent architecture with a Designer, Coder, and Reflector, reinforced by a memory archive that supports reuse and refinement of workflows. Coupled with medical coding tools and guideline-driven strategies, MedDCR provides the first domain-specific framework for automated optimisation of medical coding workflows, offering stronger performance and greater interpretability.

3 Method

3.1 Problem Formulation

We formalize automated workflow optimisation problem for medical coding as follows. Let $\mathcal{X}$ denote the space of clinical notes and $\mathcal{C}$ the set of admissible ICD codes. A dataset is

$$D = \{(x_i, Y_i)\}_{i=1}^{n}, \quad x_i \in \mathcal{X},\ Y_i \subseteq \mathcal{C}. \quad (1)$$

A *workflow* W induces a coder function

$$f_W : \mathcal{X} \to 2^{\mathcal{C}}, \quad (2)$$

that maps a note to a predicted set of codes.

Workflows are constructed from a component library $\mathcal{L}$ consisting of: (i) a tool set $\mathcal{T}$ (e.g., ICD index retrieval, guideline validators), (ii) reasoning/strategy primitives Σ (e.g., extraction, validation, reconciliation), and (iii) LLM modules $\mathcal{M}$ with parameter configurations Θ.

We represent a workflow as a directed acyclic graph

$$W = (G = (V, E), \phi), \tag{3}$$

where each node $v \in V$ instantiates a component $c_v \in \mathcal{L}$ with parameters $\phi_v \in \Theta$, and edges E define data/control flow. The *search space* of feasible workflows is

$$S = \left\{ W = (G, \phi) \;\middle|\; \begin{array}{l} G \text{ is a DAG over } \mathcal{L}, \\ W \text{ is executable} \end{array} \right\}. \tag{4}$$

Medical coding is constrained by official guidelines. Let Γ denote the guideline resource (ICD Alphabetic/Tabular Index, coding rules). A compliance oracle $V_\Gamma(W, x) \in [0, 1]$ measures the fraction of guideline checks that pass for workflow W on input x. We evaluate a workflow with a task metric $g(\cdot, \cdot)$ (e.g., micro/macro F_1) and define the objective on a validation set D_{val}:

$$\begin{aligned} G(W; D_{\text{val}}, \Gamma) = \mathbb{E}_{(x,Y)\sim D_{\text{val}}} \Big[& g(f_W(x), Y) \\ & - \lambda_{\text{viol}} \big(1 - V_\Gamma(W, x)\big) \Big] \\ & - \lambda_{\text{cost}}\, C(W), \end{aligned} \tag{5}$$

where $C(W)$ measures resource usage (e.g., tool calls, latency, tokens) and $\lambda_{\text{viol}}, \lambda_{\text{cost}} \geq 0$ control the trade-offs between predictive performance, guideline compliance, and efficiency.

The automated workflow optimisation problem for medical coding is then:

$$W^\star \in \arg\max_{W \in \mathcal{S}} \; G(W; D_{\text{val}}, \Gamma). \tag{6}$$

In the remainder of this section, we detail how MedDCR performs the iterative search for $W^\star$ guided by the objective G and the constraints Γ.

3.2 MedDCR Framework Overview

MedDCR framework instantiates the workflow optimisation problem defined in Section 3.1 by organising the search process into an iterative loop of design, coding (execution), and reflection. The framework aims to automatically discover effective workflows for medical coding without relying on fixed, manually crafted pipelines.

The loop begins with the initialisation of a memory archive $\mathcal{H}_0$, which contains a small collection of seed workflows (See Figure 1). These seeds capture both general reasoning strategies, such as chain-of-thought (Wei et al., 2022; Zhang et al., 2023), self-refinement (Madaan et al., 2023), and multi-agent debate (Du et al., 2023) and domain-specific medical coding strategies, including medical entity extractions (Douglas et al., 2025), tabular index similarity checks, and reconciliation heuristics (Gan et al., 2025), among others. The archive also stores a list of coding tools, such as alphabetic index search, parent–child code lookup, and code–description extraction modules. This initial set provides the system with a foundation of plausible behaviours, but the search is not limited to them, where the archive can expand with newly discovered strategies, including tool calls proposed dynamically by LLMs.

Once initialised, the search process proceeds iteratively through a *Design–Execute–Reflect* cycle (We provide the detailed procedure in Algorithm 1). At iteration t, a new workflow plan π_t is proposed by an LLM-based Designer agent (Li et al., 2023; Wu et al., 2024), drawing on the memory archive that contains both the seeded workflows and prior designs $\mathcal{H}_{0:t-1}$. Each plan specifies a combination of tools, strategies, and reasoning steps, as well as their execution order. The plan is then compiled

Algorithm 1 DCR Optimisation

Input: Validation set D_{val}, module library $\mathcal{L}$, objective G, iterations T, generation size M.
Output: Optimized workflow $W^\star$.
[Initialization of Archive]
Initialize memory archive $\mathcal{H}_0$ with optional, pre-evaluated seed workflows $\mathcal{W}_{\text{seed}}$.
for $t = 1$ to T **do**
    [Workflow Design Phase]
    Select elite $\mathcal{E}_t$ and recent $\mathcal{R}_t$ workflow examples from archive $\mathcal{H}_{t-1}$.
    Propose a new set of workflow plans $\{\pi_t^{(m)}\}_{m=1}^{M}$ guided by $\mathcal{E}_t$ and $\mathcal{R}_t$.
    [Code & Reflect Phase]
    Compile each plan π_t into a runnable program of the workflow W_t.
    for each workflow W_t **do**
        Execute W_t on D_{val} to obtain score s_t and reflection r_t.
        Update archive: $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \cup \{(\pi_t, W_t, s_t, r_t)\}$.
    end for
end for
[Selection of Optimal Workflow]
Obtain best workflow $W^\star \leftarrow \arg\max_{(\pi,W,s,r)\in\mathcal{H}_T} s$.
Return optimized workflow $W^\star$.

into an executable workflow W_t, translated into runnable code by a Coder agent, and subsequently executed to produce medical code predictions.

The execution results are assessed with an evaluation function $G(W_t)$, which integrates predictive performance (e.g., F1 score), guideline compliance (Ann Barta, 2009), and computational cost. Alongside the scalar score, the evaluation produces textual feedback produced by a *Reflector* agent, diagnosing workflow errors such as ineffective tool combinations or guideline violations. Both the score and feedback are appended to the archive as a record (π_t, W_t, s_t, r_t).

The memory archive thus grows with every iteration, providing two key signals for the search: (i) high-performing past workflows, which act as exemplars to imitate or refine, and (ii) diverse recent workflows, which encourage exploration. New proposals are therefore shaped both by the accumulated experience in the archive and by the evaluation feedback from prior iterations (Hu et al., 2025). Over time, the system learns to discover increasingly effective workflows by combining coding tools and strategies in novel ways.

The search loop terminates once the budget of iterations is reached or when performance converges. The final output is the best workflow $W^\star$ found in the archive, which can be used directly for automated medical coding or supplied as a strong starting point for further optimisation.

3.3 Meta-Agent Architecture

In this section, we present the details of the meta-agents within the MedDCR framework.

Designer Agent. The Designer agent is responsible for generating candidate workflow plans π. At iteration t, it outputs

$$\pi_t = \text{Designer}(\mathcal{H}_t, \mathcal{L}, \Gamma) \in \Pi, \tag{7}$$

where $\mathcal{H}_t$ is the memory archive, $\mathcal{L}$ is the component library (tools, strategies, and LLM modules), and Γ encodes coding guidelines. The Designer operates under a structured *meta-prompt* that (i) specifies constraints on how workflows must be formatted and executed, (ii) enumerates the available tool and strategy signatures, and (iii) appends informative exemplars from the memory archive.

In particular, the prompt shows the top-k best-performing workflows and the most recent n designs, ensuring that the Designer can exploit strong prior solutions while maintaining exploration.

Coder Agent. The Coder agent transforms abstract workflow plans into executable programs (e.g., Python-like codes). Given a plan π_t, it compiles the abstract plan into

$$W_t = \text{Coder}(\pi_t) \in \mathcal{S}, \tag{8}$$

where W_t is an operational program that can be executed on clinical notes to generate medical codes.

In practice, however, generated code may contain *syntax or execution errors*, which would otherwise block evaluation. To address this, the Coder incorporates a *self-fixing loop*: the executor first checks whether the compiled code runs successfully, if an error occurs, the Coder attempts to repair the code automatically and retries execution until a valid workflow is obtained or a retry budget is exceeded (Joshi et al., 2023; Zhang et al., 2024).

This mechanism ensures that the search process is not derailed by syntactic inconsistencies (Olausson et al., 2024). The separation of roles is thus preserved: the Designer explores high-level workflow planning, while the Coder guarantees that plans are translated into syntactically correct and executable implementations.

Reflector Agent. The Reflector evaluates executed workflows and provides targeted feedback. Specifically, for each workflow W_t, it outputs

$$(s_t, r_t) = \text{Reflector}(W_t, D_{\text{val}}), \tag{9}$$

where $s_t = G(W_t; D_{\text{val}}, \Gamma)$ is a scalar score that integrates predictive accuracy (e.g., F1 score or precision/recall), guideline compliance, and computational efficiency, and r_t is a textual critique.

To generate this feedback, the Reflector collects the intermediate outputs of each LLM call within the workflow and interprets them in the context of the corresponding operation. For example, it may highlight when an entity extraction step misses key mentions, when a guideline validator rejects a candidate code, or when reconciliation produces redundant outputs. The tuple (π_t, W_t, s_t, r_t) is then appended to the archive, which is updated as

$$\mathcal{H}_{t+1} = \mathcal{H}_t \cup \{(\pi_t, W_t, s_t, r_t)\}. \tag{10}$$

This combination of quantitative scoring (Khattab et al., 2024) and fine-grained textual feedback (Yang et al., 2023a; Pryzant et al., 2023) ensures that subsequent iterations improve not only raw performance but also the robustness and interpretability of workflows.

Method Category (Label Space)	Model	MDACE Precision	MDACE Recall	MDACE F1	ACI-BENCH Precision	ACI-BENCH Recall	ACI-BENCH F1
PLM (1K)	ICD (Huang et al., 2022)	**0.49**	0.47	**0.48**	0.43	0.41	0.42
	CA (Douglas et al., 2025)	**0.46**	0.45	0.45	**0.44**	0.42	0.43
Coder Workflow (1K)	RRS (Kwan, 2024)	0.24	0.30	0.27	0.26	0.52	0.35
	MAC (Li et al., 2024b)	0.27	0.31	0.29	0.23	0.50	0.31
	CLH (Motzfeldt et al., 2025)	0.45	0.40	0.42	**0.44**	0.39	0.41
Agentic Method (1K)	CoT (Wei et al., 2022)	0.30	0.31	0.30	0.35	0.50	0.41
	CoT-SC (Wang et al., 2023a)	0.39	0.43	0.41	0.36	0.59	0.44
	MulDe (Du et al., 2023)	0.21	0.40	0.28	0.16	0.65	0.25
	Judge (Zheng et al., 2023)	0.26	0.53	0.35	0.22	0.64	0.33
	MNER (Goel, 2025)	0.13	0.48	0.20	0.15	**0.67**	0.25
	ADAS (Hu et al., 2025)	0.37	0.51	0.43	0.28	0.59	0.43
	MedDCR-GPT-4o	0.41	**0.55**	0.47	0.36	0.65	**0.46**
	MedDCR-GPT-5	**0.46**	**0.59**	**0.51**	**0.43**	**0.67**	**0.52**

Table 1: **Main results on MDACE and ACI-BENCH datasets.** The best results are highlighted in bold, and the second-best results are shown in **gray bold**. Methods are grouped into three categories: Pretrained Language Models, expert-designed coding workflows, and agentic workflow strategies (including agent-based search methods).

3.4 Memory Archive and Plug-and-Play

A central component of our proposed framework is the memory archive, which records the history of all explored workflows. At iteration t, the archive $\mathcal{H}_t$ contains tuples

$$\mathcal{H}_t = \{(\pi_j, W_j, s_j, r_j)\}_{j<t}, \tag{11}$$

where π_j is the workflow plan proposed by the Designer, W_j is the compiled executable workflow, s_j is the score, and r_j is the textual feedback.

This archive plays a dual role: it enables *reuse* of effective workflows by presenting the top-k highest scoring exemplars to the Designer, and it supports *exploration* by also presenting the n most recent workflows, which prevent the search from collapsing into repetitive local optima (Zhu et al., 2023).

In this way, the Designer is encouraged to learn from successful past patterns while avoiding premature convergence to a single family of workflows (Zhong et al., 2024). As the search progresses, the archive grows into a structured memory of workflow designs, making it an adaptive optimiser that improves continuously over time.

Beyond storing internal proposals, the memory archive also supports a *plug-and-play* mode. Here, external expert-crafted workflows, denoted $\mathcal{W}_{\text{seed}}$, are injected into the initial archive $\mathcal{H}_0$. These workflows may come from prior research, human-designed pipelines, or established clinical coding heuristics (Gan et al., 2025; Douglas et al., 2025; Motzfeldt et al., 2025). Once seeded, MedDCR treats them like any other entry in the archive: they can be directly reused, refined through feedback, or combined with newly generated strategies.

This property makes our framework workflow-agnostic: it can either discover new workflows from scratch or improve upon existing designs, depending on the application scenario. The final output is therefore not limited to the best design discovered during search, but can also include optimised variants of expert workflows, providing a bridge between automated search and human expertise.

4 Experiments

4.1 Experimental Setup

Dataset. We evaluate our framework on two publicly available ICD-10 coding benchmarks. MDACE (Cheng et al., 2023) provides expert-verified ICD-10 annotations to a mix of inpatient and professional-fee charts drawn from MIMIC-III (Johnson et al., 2016), including discharge summaries, radiology reports and physician notes. ACI Benchmark (Yim et al., 2023; Baksi et al., 2025) is a synthetic dataset of clinical notes paired with ICD-10 codes for automated coding systems. Together, they enable evaluation of both coding accuracy and explainability in real-world settings.

Baseline. We benchmark MedDCR against a broad set of existing approaches spanning *three* paradigms. First, we consider pre-trained language model methods that frame coding as multi-label classification and require re-training on coding datasets, including PLM-ICD(Huang et al.,

Method	MDACE			ACI-BENCH		
	Search Tokens	Exec. Tokens	Cost (USD)	Search Tokens	Exec. Tokens	Cost (USD)
MedDCR	19,994	11,545,586	**$17.09**	14,294	2,233,623	**$5.72**

Table 2: **Computation Cost Comparison.** Token usage and projected cost in USD per 100 inference samples per search loop on MDACE and ACI-BENCH datasets. While effective, our method remains cost-efficient.

Seed Setting	MDACE		
	Precision	Recall	Micro-F1
CoT-SC (seed only)	0.37	0.48	0.42
+ Quality-Diversity	0.38	0.48	0.42
+ Multi-Debate	0.37	0.53	0.44
+ NER	0.35	0.55	0.43
+ All auxiliary	**0.41**	**0.55**	**0.47**
No seed (scratch)	0.39	0.51	0.44

Table 3: **Plug-and-play validation**: MedDCR initialised with CoT-SC as the primary seed, optionally augmented with auxiliary seeds (Self-Refine, Multi-Debate, Med-NER). Optimisation consistently improves the baseline CoT-SC workflow and outperforms search from scratch.

	Precision	Recall	Micro-F1
MedDCR-GPT-4o	0.41	0.55	0.47
– No Top-k	0.35	0.42	0.38
– No Recent-n	0.33	0.34	0.34
– No Score Feedback	0.39	0.47	0.43
– No Text Feedback	0.44	0.50	0.47
– No Guideline in Meta	0.40	0.57	0.47
– No Exemplar in Meta	0.36	0.51	0.42

Table 4: **Ablation study** on MDACE dataset. Each row removes one core component of MedDCR. The performance drops confirm the importance of the workflow exemplars, reflector feedback, guideline constraints and few-shot exemplar in achieving the full performance.

2022), and PLM-CA (Douglas et al., 2025). Second, we include expert-designed medical workflows that rely on heuristic rules or structured coding pipelines, represented by RRS (Kwan, 2024), MAC (Li et al., 2024b), and CLH (Motzfeldt et al., 2025). Finally, we evaluate against general agentic strategies and automated search frameworks that build on large language models with structured reasoning or multi-agent coordination, such as Chain-of-Thought (Wei et al., 2022), Self-Consistency (Wang et al., 2023b), Multi-Debate (Du et al., 2023), Self-Refine (Shinn et al., 2023), and NER (Goel, 2025), as well as recent optimisation frameworks like ADAS (Hu et al., 2025). GPT-4o (Hurst et al., 2024) serves as the backbone model for all baseline agents. Together, these baselines span model-based, rule-based, and agentic paradigms, enabling a comprehensive comparison.

Implementations. We use GPT-based models as the backbone for MedDCR, evaluating both GPT-4o (Hurst et al., 2024) and GPT-5 (OpenAI, 2025). The search loop is run for 100 iterations, with the Designer conditioned on the top-5 highest scoring and 3 most recent workflows from the archive at each step. To initialise the archive, we include all baselines from the aforementioned agentic workflow group, ensuring a diverse starting pool. Coding tools are modified and adapted from the simple-icd-10 library, covering both ICD-10-CM and PCS codes. More details are in the Appendix D.

Evaluation Metrics. We report standard measures for extreme multi-label coding: micro-F1, precision, and recall (Huang et al., 2022; Edin et al., 2024; Douglas et al., 2025). Micro-F1 balances overall precision and recall across the large code space, providing a primary measure of workflow effectiveness. Precision reflects the system's ability to avoid spurious code assignments, while recall captures its capacity to recover the full set of relevant codes. Together, these metrics indicate how well the proposed workflows navigate the trade-off between exploration (capturing diverse correct codes) and exploitation (maintaining accuracy).

4.2 Main Results

As shown in Table 1, MedDCR consistently outperforms all baselines on both MDACE and ACI-BENCH. On MDACE, MedDCR-GPT-5 achieves a Micro-F1 of 0.51, improving over the strongest PLM baseline (ICD) and the best agentic baseline (CoT-SC) by 6%. On ACI-BENCH, MedDCR-GPT-5 reaches 0.52 for F1, again surpassing PLM and agentic baselines. These results demonstrate that automated workflow search yields more effective coding strategies than either retrained language models or expert-crafted agentic pipelines. Importantly, Table 2 demonstrates that these improvements are achieved with modest overhead. Despite the iterative search process, MedDCR remains cost-

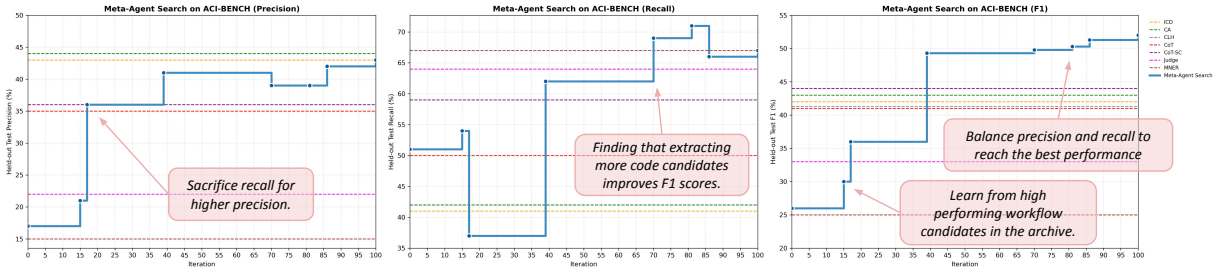

Figure 2: **Case study of the search process on ACI-Bench.** The blue line tracks the best workflow discovered at each iteration, measured by F1. The figure illustrates how performance improves as the system explores diverse candidates, learns from high-performing workflows, and balances precision and recall to refine the final design.

efficient: a single search loop requires fewer than 20k search tokens on MDACE and 15k on ACI-BENCH, with total costs of $17.09 and $5.72 per 100 inference samples, respectively. Notably, the vast majority of cost arises from execution tokens rather than search tokens, showing that workflow optimisation overhead is modest relative to workflow execution. Finally, Table 3 validates MedDCR's plug-and-play property. Using CoT-SC as the primary seed, optimisation alone improves F1 from 0.41 to 0.42. When augmented with auxiliary seeds, performance increases further, reaching 0.47 F1 with all auxiliaries. Compared to search from scratch, seeded optimisation also achieves higher final scores. This confirms that MedDCR can flexibly refine and use existing workflows and diverse strategies, making it practical as a general workflow optimiser for medical coding.

4.3 Ablation Studies

Table 4 reports ablations on MDACE, where we systematically remove key components of MedDCR. Removing the top-k or recent-n workflows causes the largest performance drops of 0.09 and 0.13 in F1, confirming that both exploitation of strong designs and diversity from recent ones are essential for effective search. Feedback from the reflector is also critical: without score feedback, F1 falls to 0.43, while removing textual feedback prevents further improvements despite retaining numeric scores. Guideline constraints show a trade-off, slightly increasing recall but offering no net F1 gain when excluded. Finally, removing all exemplars from the meta-prompt reduces F1 to 0.42, underscoring their role in stabilizing the workflow generation. Overall, the results highlight that MedDCR's gains arise from the complementary contributions of memory selection, reflective feedback, guideline grounding, and exemplar conditioning.

4.4 Case Study

Figure 2 illustrates how our framework improves over the course of the search in terms of precision, recall, and F1, with the blue line indicating the best workflow discovered at each iteration as measured by F1. In the early stages, the search explores relatively simple strategies, while later iterations integrate more complex tool use and validation steps, resulting in steady performance gains. This progression highlights the role of memory and reflective feedback in escaping suboptimal designs and converging toward stronger workflows. The final best workflow combines multiple reasoning strategies and coding tools in a coordinated sequence, exemplifying the kind of designs that MedDCR can automatically uncover. Due to space, the best workflow structure is provided in the Appendix C.

5 Conclusions

We presented MedDCR, an automated framework for workflow optimisation in medical coding. MedDCR treats agentic workflow construction as a learning problem, realised through iterative design–execute–reflect loops. Central to the framework is a meta-agent architecture with a Designer, Coder, and Reflector, supported by a memory archive that enables both search from scratch and plug-and-play refinement of expert workflows. Experiments on MDACE and ACI-BENCH demonstrate that MedDCR achieves consistent improvements over state-of-the-art baselines, while also providing interpretable feedback through the Reflector. These results show that automated workflow optimisation can yield both higher accuracy and greater reliability than static, manually designed systems, offering a principled approach to building trustworthy medical coding agents.

6 Limitations

While MedDCR demonstrates strong performance and adaptability, several limitations remain. First, the framework relies on general-purpose GPT models that are not fine-tuned for medical coding, which may limit their understanding of domain-specific terminology and guideline nuances. A medically fine-tuned or instruction-adapted foundation model could further improve both accuracy and compliance with coding rules. Second, workflow optimisation is computationally expensive, as it involves repeated LLM executions and feedback loops. Although the search cost is moderate relative to inference, scaling to larger datasets or real-time settings may require more efficient search strategies. Third, the textual feedback generated by the Reflector is heuristic and may occasionally produce ambiguous or overly general guidance, suggesting a need for structured reflection or rule-grounded critics. Finally, our experiments focus primarily on ICD-10 coding, extending MedDCR to multi-taxonomy or multi-lingual settings would provide stronger evidence of generality. Addressing these limitations would bring MedDCR closer to deployment-ready, domain-specialised agentic systems for clinical applications.

References

Vera Alonso, João Vasco Santos, Marta Pinto, Joana Ferreira, Isabel Lema, Fernando Lopes, and Alberto Freitas. 2020. Problems and barriers during the process of clinical coding: a focus group study of coders' perceptions. *Journal of medical systems*, 44(3):62.

MSA Ann Barta. 2009. Icd-10-cm official coding guidelines. *Journal of AHIMA*.

Krishanu Das Baksi, Elijah Soba, John J Higgins, Ravi Saini, Jaden Wood, Jane Cook, Jack I Scott, Nirmala Pudota, Tim Weninger, Edward Bowen, and 1 others. 2025. Medcoder: A generative ai assistant for medical coding. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pages 449–459.

Joseph Spartacus Boyle, Antanas Kascenas, Pat Lok, Maria Liakata, and Alison Q O'Neil. 2023. Automated clinical coding using off-the-shelf large language models. In *Deep Generative Models for Health Workshop, Advances in neural information processing systems*.

Sharon Campbell and Katrina Giadresco. 2020. Computer-assisted clinical coding: A narrative review of the literature on its benefits, limitations, implementation and impact on clinical coding professionals. *Health Information Management Journal*, 49(1):5–18.

Pengfei Cao, Chenwei Yan, Xiangling Fu, Yubo Chen, Kang Liu, Jun Zhao, Shengping Liu, and Weifeng Chong. 2020. Clinical-coder: Assigning interpretable icd-10 codes to chinese clinical notes. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 294–301.

Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. 2023. Unleashing the potential of prompt engineering in large language models: a comprehensive review. *arXiv preprint arXiv:2310.14735*.

Hua Cheng, Rana Jafari, April Russell, Russell Klopfer, Edmond Lu, Benjamin Striner, and Matthew Gormley. 2023. MDACE: MIMIC documents annotated with code evidence. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7534–7550.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2024. Chain-of-verification reduces hallucination in large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3563–3578.

Hang Dong, Matúš Falis, William Whiteley, Beatrice Alex, Joshua Matterson, Shaoxiong Ji, Jiaoyan Chen, and Honghan Wu. 2022. Automated clinical coding: what, why, and where we are? *NPJ digital medicine*, 5(1):159.

Hang Dong, Víctor Suárez-Paniagua, William Whiteley, and Honghan Wu. 2021. Explainable automated coding of clinical notes using hierarchical label-wise attention networks and label embedding initialisation. *Journal of biomedical informatics*, 116:103728.

James C Douglas, Yidong Gan, Ben Hachey, and Jonathan K Kummerfeld. 2025. Less is more: Explainable and efficient icd code prediction with clinical entities. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30835–30847.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*.

Joakim Edin, Maria Maistro, Lars Maaløe, Lasse Borgholt, Jakob Havtorn, and Tuukka Ruotsalo. 2024. An unsupervised approach to achieve supervised-level explainability in healthcare records. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4869–4890.

Matúš Falis, Aryo Pradipta Gema, Hang Dong, Luke Daines, Siddharth Basetti, Michael Holder, Rose S Penfold, Alexandra Birch, and Beatrice Alex. 2024. Can gpt-3.5 generate and code discharge summaries? *Journal of the American Medical Informatics Association*, 31(10):2284–2293.

Richárd Farkas and György Szarvas. 2008. Automatic construction of rule-based icd-9-cm coding systems. *BMC bioinformatics*, 9(Suppl 3):S10.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Forty-first International Conference on Machine Learning*.

Yidong Gan, Maciej Rybinski, Ben Hachey, and Jonathan K. Kummerfeld. 2025. Aligning AI research with the needs of clinical coding workflows: Eight recommendations based on US data analysis and critical review. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 909–922.

Zelalem Gero, Chandan Singh, Hao Cheng, Tristan Naumann, Michel Galley, Jianfeng Gao, and Hoifung Poon. 2023. Self-verification improves few-shot clinical information extraction. In *ICML 3rd Workshop on Interpretable Machine Learning in Healthcare (IMLH)*.

Akshay Goel. 2025. LangExtract.

Shengran Hu, Cong Lu, and Jeff Clune. 2025. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*.

Chao-Wei Huang, Shang-Chi Tsai, and Yun-Nung Chen. 2022. Plm-icd: Automatic icd coding with pretrained language models. *ClinicalNLP 2022*, page 10.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Shaoxiong Ji, Xiaobo Li, Wei Sun, Hang Dong, Ara Taalas, Yijia Zhang, Honghan Wu, Esa Pitkänen, and Pekka Marttinen. 2024. A unified review of deep learning for automated medical coding. *ACM Computing Surveys*, 56(12):1–41.

Alistair Johnson, Lucas Bulgarelli, Tom Pollard, Steven Horng, Leo Anthony Celi, and Roger Mark. 2020. Mimic-iv. *PhysioNet. Available online at: https://physionet. org/content/mimiciv/1.0/(accessed August 23, 2021)*, pages 49–55.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9.

Harshit Joshi, José Cambronero Sanchez, Sumit Gulwani, Vu Le, Gust Verbruggen, and Ivan Radiček. 2023. Repair is nearly generation: Multilingual program repair with llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5131–5140.

Ramakanth Kavuluru, Anthony Rios, and Yuan Lu. 2015. An empirical evaluation of supervised learning approaches in assigning diagnosis codes to electronic medical records. *Artificial intelligence in medicine*, 65(2):155–166.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, and 1 others. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*.

Keith Kwan. 2024. Large language models are good medical coders, if provided with tools. *arXiv preprint arXiv:2407.12849*.

Fei Li and Hong Yu. 2020. Icd coding from clinical text using multi-filter residual convolutional neural network. In *proceedings of the AAAI conference on artificial intelligence*, pages 8180–8187.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Junyou Li, Qin Zhang, Yangbin Yu, QIANG FU, and Deheng Ye. 2024a. More agents is all you need. *Transactions on Machine Learning Research*.

Rumeng Li, Xun Wang, and Hong Yu. 2024b. Exploring llm multi-agents for icd coding. *arXiv preprint arXiv:2406.15363*.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17889–17904.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Andreas Motzfeldt, Joakim Edin, Casper L Christensen, Christian Hardmeier, Lars Maaløe, and Anna Rogers. 2025. Code like humans: A multi-agent solution for medical coding. *arXiv preprint arXiv:2509.05378*.

James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. 2018. Explainable prediction of medical codes from clinical text. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1101–1111.

Akram Mustafa, Usman Naseem, and Mostafa Rahimi Azghadi. 2025. Evaluating hierarchical clinical document classification using reasoning-based llms. *arXiv preprint arXiv:2507.03001*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations*.

OpenAI. 2025. [link].

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Murray Shanahan, Kyle McDonell, and Laria Reynolds. 2023. Role play with large language models. *Nature*, 623(7987):493–498.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Thanh Vu, Dat Quoc Nguyen, and Anthony Nguyen. 2021. A label attention model for icd coding from clinical text. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3335–3341.

Wenxiao Wang, Priyatham Kattakinda, and Soheil Feizi. 2025. Maestro: Joint graph & config optimization for reliable ai agents. *arXiv preprint arXiv:2509.04642*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023a. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023a. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Zhichao Yang, Sanjit Singh Batra, Joel Stremmel, and Eran Halperin. 2023b. Surpassing gpt-4 medical coding with a two-stage approach. *arXiv preprint arXiv:2311.13735*.

Zhichao Yang, Sunjae Kwon, Zonghai Yao, and Hong Yu. 2023c. Multi-label few-shot icd coding as autoregressive generation with prompt. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5366–5374.

Wen-wai Yim, Yujuan Fu, Asma Ben Abacha, Neal Snider, Thomas Lin, and Meliha Yetisgen. 2023. Aci-bench: a novel ambient clinical intelligence dataset for benchmarking automatic visit note generation. *Nature Scientific Data*.

Zheng Yuan, Chuanqi Tan, and Songfang Huang. 2022. Code synonyms do matter: Multiple synonyms matching network for automatic icd coding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 808–814.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, LEI BAI, and Xiang Wang. 2025a. Multi-agent architecture search via agentic supernet. In *Forty-second International Conference on Machine Learning*.

Jialu Zhang, José Pablo Cambronero, Sumit Gulwani, Vu Le, Ruzica Piskac, Gustavo Soares, and Gust Verbruggen. 2024. Pydex: Repairing bugs in introductory python assignments using llms. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):1100–1124.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025b. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2023. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19724–19731.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Ö Arık. 2025. Multi-agent design: Optimizing agents with better prompts and topologies. *arXiv preprint arXiv:2502.02533*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, and 1 others. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

Appendix

A Medical Coding Background

Medical coding task. Medical coding is the process of translating unstructured clinical documentation into standardized diagnostic and procedural codes, most commonly following the International Classification of Diseases (ICD) standard. Each clinical note may contain multiple diagnoses, procedures, and contextual information that must be mapped to corresponding ICD-10-CM (diagnosis) and ICD-10-PCS (procedure) codes. The task is inherently *multi-label* and *context-dependent*.

Example. Consider the following excerpt from a fabricated discharge summary:

> *"The patient was admitted for acute myocardial infarction and underwent coronary artery bypass graft (CABG). The postoperative course was uncomplicated."*

A correct ICD-10-CM coding outcome for this note includes:

- `I21.3` — ST elevation (STEMI) of unspecified site.

and for ICD-10-PCS (procedure codes):

- `021009W` — Bypass coronary artery, one site, autologous vein, open approach.

Challenges. Accurate code assignment requires multiple reasoning steps:

1. Identifying relevant clinical entities (e.g., diseases, procedures).
2. Consulting external resources such as the ICD alphabetic index and tabular list.
3. Applying coding guidelines (e.g., sequencing, laterality, and exclusion rules).
4. Cross-checking consistency between diagnoses and procedures.

For example, a note describing "Type 2 diabetes with chronic kidney disease" must yield both the primary diagnosis (`E11.22`) and the complication (`N18.9`) while respecting guideline-specific linkage rules.

Motivation for automation. Given the need for cross-referencing multiple resources and enforcing rule-based logic, medical coding naturally lends itself to *workflow-based* reasoning rather than direct text classification. The MedDCR framework aims to automate this multi-step reasoning process by discovering workflows that can effectively combine tool use, rule application, and reflection.

B Data Consent and Usage

All datasets used in this study are derived from publicly available, de-identified clinical corpora with appropriate data use agreements. The MDACE dataset (Cheng et al., 2023) is constructed from the MIMIC-III database (Johnson et al., 2016), which contains de-identified electronic health records from the Beth Israel Deaconess Medical Center.

Access to MIMIC-III requires completion of the PhysioNet credentialing process and acceptance of a data use agreement that prohibits any attempt at patient re-identification. Our use of MDACE fully complies with these requirements.

The ACI-BENCH dataset (Yim et al., 2023) is publicly released under an open-access license and contains synthetic or anonymized clinical notes for benchmarking purposes. No protected health information (PHI) was accessed or processed in any part of this study. All experiments were conducted in accordance with institutional data governance and ethical use policies.

C Case Study and Pseudo-Code of the Searched Workflow

Overview. To better illustrate how MedDCR operates in practice, we present the best workflows discovered on the ACI-Benchmark dataset. This case study corresponds to the pointed workflow in Figure 2, showing how iterative search progressively refines design choices through reflective feedback and archive-guided learning (Algorithm 2 and 3).

Workflow evolution. In early iterations, the Designer generated linear pipelines focused mainly on entity extraction and description matching. As search progressed, reflective feedback encouraged the integration of verification and reconciliation steps, such as cross-checking alphabetic and tabular index results or re-validating low-confidence predictions. The final workflow incorporates diagnostic reasoning, combining multiple coding tools with validation and guideline enforcement to ensure consistency for the final predictions.

Pseudo-code of the searched workflow. The pipeline balances recall and precision through a two-stage process. In the first stage (Lines 1–7, Algo 2), it expands recall by generating a large pool of candidate codes from multiple sources including alphabetic index lookup with extracted terms (expanded with synonyms), term-based, and note-based proposals from LLMs, ensuring broad coverage. In the second stage, precision is progressively enforced through validation, evidence linking, and filtering. Invalid codes are removed via rule checks, an evidence-aware judge filters low-confidence candidates based on τ_{keep}, and contrastive screening γ prunes near-duplicates. Finally, reranking integrates judge scores, description similarity, and evidence strength to prioritize high-precision codes while maintaining recall from the initial expansion.

Algorithm 2 Evidence-Aware Coding Pipeline

Require: Clinical note N; hyperparameters: T_{terms} (term extractor), $S{=}4$, $S_{\max}{=}2$, $W{=}24$, $\tau_{\text{keep}}{=}0.5$, $\gamma{=}0.15$, $K{=}20$

Ensure: Ranked list $\mathcal{R}$ of codes with scores and evidence

1: $E \leftarrow \textsc{Terms}(\textsc{ReasonForVisit}(N); T_{\text{terms}})$ ▷ extract terms/entities w/ reason-for-visit cues
2: $C_s \leftarrow \textsc{SearchAlphaIndex}(\textsc{SynExp}(E))$
3: $C_t \leftarrow \textsc{ProposeFromTerms}(E; \text{mode} = \text{plain})$
4: $C_{tc} \leftarrow \textsc{ProposeFromTermsCoT}(E; \text{ref} = \text{sec}; \text{mode} = \text{CoT})$
5: $C_n \leftarrow \textsc{ProposeFromNote}(\text{RFV}(N); \text{mode} = \text{plain}; \text{samples} = S)$
6: $C_{nc} \leftarrow \textsc{ProposeFromSec}(N; \text{mode} = \text{CoT}, \text{samples} = S)$
7: $\hat{C} \leftarrow \textsc{Merge}(C_s, C_t, C_n, C_{tc}, C_{nc})$ ▷ set union with deduplication
8: $\hat{C} \leftarrow \textsc{Canonicalize}(\hat{C})$
9: $\hat{C} \leftarrow \textsc{Validate}(\hat{C})$ ▷ schema/rule compliance; remove invalid codes
10: $D \leftarrow \textsc{FetchDesc}(\hat{C})$ ▷ D : map code $\mapsto$ description from tabular index
11: $m_{\text{desc}} \leftarrow \textsc{DescMatch}(\hat{C}, D, N)$ ▷ per-code description$\leftrightarrow$note match score
12: $Z \leftarrow \textsc{EvidenceLink}(N, \hat{C}; S_{\max}, W)$ ▷ Z : map code $\mapsto$ at most $S_{\max}$ snippets (window W tokens)
13: $S_{\text{judge}} \leftarrow \textsc{Judge}(\hat{C}, Z;$ judge_strategy $=$ per-code evidence-aware keep/drop, output $=$ json_scores) ▷ score in $[0, 1]$ for each code
14: $\tilde{C} \leftarrow \{\, c \in \hat{C} \;:\; S_{\text{judge}}(c) \geq \tau_{\text{keep}} \,\}$ ▷ filter by judge score
15: $\tilde{C} \leftarrow \textsc{ContrastiveScreen}(\tilde{C};$ by $=$ sibling_desc, contrast_margin $= \gamma$, action $=$ drop_or_demote) ▷ prune/demote near-duplicate siblings with small description margin
16: **for all** $c \in \tilde{C}$ **do**
$s(c) \leftarrow \textsc{RerankScore}(S_{\text{judge}}(c), m_{\text{desc}}(c), \textsc{EvidenceStrength}(Z(c)))$
17: **end for**
18: $\mathcal{R} \leftarrow \textsc{SortBy}(\tilde{C},$ key $= s,$ desc$)$
19: **if** $|\mathcal{R}| < K$ **then**
20: $\mathcal{B} \leftarrow \textsc{FallbackTopK}(\hat{C} \setminus \tilde{C},\ K - |\mathcal{R}|)$
21: $\mathcal{R} \leftarrow \textsc{Concat}(\mathcal{R},\ \mathcal{B})$
22: **end if**
23: **return** $\textsc{ReturnRaw}(\mathcal{R},\ s,\ Z,\ m_{\text{desc}})$

Algorithm 3 Evidence- and Vote-Aware Coding

Require: Clinical note N; S_t=3, S_n=4, $S_{\max}$=2, M=256, θ=0.5, K=20

Ensure: Ranked list $\mathcal{R}$ of codes with scores and evidence

$E \leftarrow \textsc{Terms}(N)$

$C_t^{(1:S_t)} \leftarrow \textsc{ProposeFromTerms}(E;\ \text{mode} = \text{plain},\ \text{samples} = S_t)$

$C_n^{(1:S_n)} \leftarrow \textsc{ProposeFromNote}(N;\ \text{mode} = \text{CoT},\ \text{samples} = S_n)$

$\hat{C} \leftarrow \textsc{Merge}\Big(\bigcup_{s=1}^{S_t} C_t^{(s)},\ \bigcup_{s=1}^{S_n} C_n^{(s)}\Big)$

$\hat{C} \leftarrow \textsc{Canonicalize}(\hat{C})$

$\hat{C} \leftarrow \textsc{Validate}(\hat{C})$ ▷ Self-consistency via normalized vote frequency

$\text{votes}(c) \leftarrow \sum_{s=1}^{S_t} \mathbf{1}\{c \in C_t^{(s)}\} + \sum_{s=1}^{S_n} \mathbf{1}\{c \in C_n^{(s)}\}$ for $c \in \hat{C}$

$\text{vote\_ratio}(c) \leftarrow \dfrac{\text{votes}(c)}{S_t + S_n}$; annotate as "vote_ratio"

$D \leftarrow \textsc{FetchDesc}(\hat{C})$ ▷ D: code $\mapsto$ tabular description

$m_{\text{desc}} \leftarrow \textsc{DescMatch}(\hat{C}, D, N)$ ▷ per-code description$\leftrightarrow$note match score in $[0, 1]$

$Z \leftarrow \textsc{EvidenceExtract}(N, \hat{C};\ \text{strategy} = \text{snippet\_from\_note},\ \text{per\_code\_spans} = S_{\max},\ \text{max\_tokens} = M)$

$\text{evidence\_overlap}(c) \leftarrow \textsc{EvidenceOverlap}(Z(c), N)$; annotate as "evidence_overlap"

$(\text{judge\_keep}(c),\ \text{judge\_conf}(c)) \leftarrow \textsc{Judge}(c, Z(c), D(c);\ \text{strategy} = \text{evidence\_tabular\_desc})$ for $c \in \hat{C}$

annotate judge_conf as "judge_conf"

$\tilde{C} \leftarrow \{\, c \in \hat{C} : \text{judge\_keep}(c) \vee \text{judge\_conf}(c) \geq \theta \,\}$

$\tilde{C} \leftarrow \textsc{HierPrune}(\tilde{C};\ \text{rules} = \{\text{prefer\_specific\_over\_unspecified}, \text{drop\_duplicate\_laterality}, \text{drop\_mutually\_exclusive\_with\_lower\_conf}\})$

for all $c \in \tilde{C}$ **do**

  $s(c) \leftarrow 0.4 \cdot \text{vote\_ratio}(c) + 0.3 \cdot m_{\text{desc}}(c) + 0.2 \cdot \text{judge\_conf}(c) + 0.1 \cdot \text{evidence\_overlap}(c)$

end for

$\mathcal{R} \leftarrow \textsc{SortBy}(\tilde{C},\ \text{key} = s,\ \text{desc})$

if $|\mathcal{R}| < K$ **then**

  $\mathcal{B} \leftarrow \textsc{FallbackTopK}(\hat{C} \setminus \tilde{C},\ K - |\mathcal{R}|)$

  $\mathcal{R} \leftarrow \textsc{Concat}(\mathcal{R},\ \mathcal{B})$

end if

return $\textsc{ReturnRaw}(\mathcal{R},\ s,\ Z,\ \text{vote\_ratio},\ m_{\text{desc}},\ \text{judge\_conf},\ \text{evidence\_overlap})$

D Meta-Prompt and Coding Tools

Meta-Prompt for the Designer Agent

The Designer agent is instructed through a meta-prompt that defines its role, input exemplars, design constraints, and expected JSON output format. The prompt dynamically incorporates the top-k best-performing and n most-recent workflows from the memory archive. Below is the formatted version of the prompt used in this study.

Listing 1: Meta-prompt of Designer Agent

```
You are designing a NEW controller for ICD-10 multi-label
    ↪ medical coding.
STAGE A (THIS TURN): PROPOSE A PLAN ONLY - DO NOT WRITE
    ↪ CODE.
Return a compact JSON plan that the system will implement
    ↪ later.

Think creatively while maintaining coding accuracy.

Rules:
- The plan is an ordered list of steps; each step is an
    ↪ object with an operation key ("op")
  and its required parameters.
- Exclude codes related to family history, negative,
    ↪ hypothetical, or ruled-out mentions.
- Include key parameters when relevant (e.g., samples,
    ↪ threshold, k, strategy, by).
- Avoid repeating recent workflow families unless changes
    ↪ are expected to improve F1.
- Keep the workflow concise and executable; the system will
    ↪ validate it.
- You may combine or extend existing operations if
    ↪ logically beneficial.

Available Tool Signatures:
MedicalTermExtraction(note) - Extract medical terms from
    ↪ the medical notes using LangExtract.
TabularIndexSearch(entity)   - Retrive code description
    ↪ from tabular index.
DescMatch(codes, note)       - Compute similarity between
    ↪ code descriptions and note text.
GuidelineValidator(codes, ruleset) - Filter codes violating
    ↪ ICD-10 rules.
Reconciler(codes)                - Merge duplicates or
    ↪ conflicting predictions.
EvidenceLinker(note, codes, window, max_snips) - Extract
    ↪ evidence snippets per code.
Judge(codes, evidence, strategy)  - Assign per-code
    ↪ confidence scores.
...

Exemplars (for diversity & performance):
First are top performers by F1 (TOP_i), followed by the
    ↪ most recent attempts (RECENT_i).
Use these to inspire but not copy - propose an improved or
    ↪ novel workflow.

Top_1:
[Exemplar Workflow i]
...

Recent_1:
[Exemplar Workflow j]
...

Deliverable (Stage A): Return STRICTLY ONE JSON OBJECT (no
    ↪ prose):
{
  "name": "<controller name>",
  "thought": "<why this plan should improve F1>",
  "plan": [ { "op": "...", "...": "..." }, ... ]
}
If you introduce new operations, name them clearly - the
    ↪ system will map or extend the op set.
DO NOT include executable code in this turn.
```

This meta-prompt encourages creativity while

constraining the Designer to produce executable, guideline-compliant workflow plans. Tool signatures are embedded to ensure awareness of available operations and their expected parameters.

Meta-Prompt for the Coder Agent

The Coder agent translates workflow plans from the Designer into executable Python code that implements the proposed controller. The prompt strictly enforces the plan order, ensures syntactic validity, and outputs JSON-formatted code only. The full formatted prompt is shown below.

Listing 2: Meta-prompt of Coder Agent

```
Stage B: Implement code that EXACTLY follows the accepted
      ↪ plan.

Implement ‘class Controller‘ for ICD-10 multi-label coding
      ↪ by following this accepted plan JSON:
<INSERT PLAN JSON HERE>

Constraints:
 - Keep EXACT signature:
       class Controller:
           def forward(self, task)
 - Honour the plan order and parameters (thresholds,
     ↪ temperatures, samples, k, strategies, etc.).
 - Do NOT include any comments in code.
 - Return STRICT JSON ONLY (no prose, no markdown):
   {
     "code": "class Controller:\n    def forward(self,
     ↪ task):\n        ..."
   }

Implementation guidance:
 - For LLM calls, use LLMAgentBase.run_text with explicit
     ↪ roles and keep temperature/loops consistent with
     ↪ the plan.
 - Always de-duplicate outputs.

 === REFERENCE EXAMPLES (for guidance only; DO NOT change
     ↪ your plan) ===
 Reference plan JSON:
   <example plan JSON>

 Reference full implementation (follows the reference plan):
   <example implementation code>

 Additional example snippets:
   <short code fragments for guidance>
```

The Coder's self-fixing loop detects non-executable code via Python traceback parsing and regenerates corrected versions until a valid workflow is obtained.

Meta-Prompt for the Reflector Agent

The Reflector agent evaluates workflow executions, producing both quantitative scores and textual feedback. It receives the workflow plan, predictions, and gold codes, and is responsible for scoring and analysing workflow effectiveness. The structured prompt used is shown below.

Listing 3: Meta-prompt of Reflector Agent

```
You are the Reflector agent in the MedDCR framework.

Your goal is to evaluate the performance of a medical
      ↪ coding workflow
and provide concise feedback that helps the Designer
      ↪ improve in the next iteration.

Inputs:
 - Workflow plan JSON describing the sequence of operations.
 - Model predictions with supporting evidence spans.
 - Ground-truth ICD-10 codes from the validation set.
 - Quantitative scores (precision, recall, F1) computed for
      ↪ this workflow.

Tasks:
 1. Verify the provided scores for correctness and internal
      ↪ consistency.
 2. Identify the workflow's strengths and weaknesses in
      ↪ terms of tool use,
    reasoning order, and guideline adherence.
 3. Analyse failure cases such as:
      * incorrect entity extraction,
      * guideline violations,
      * missing or redundant validation steps,
      * low-confidence or conflicting predictions.
 4. Provide concise textual feedback that diagnoses the
      ↪ cause of errors
    and suggests actionable refinements (e.g., add
      ↪ validation, change order of tools, adjust
      ↪ threshold).
 5. Maintain objectivity-do not modify the workflow plan or
      ↪ fabricate new scores.

Return STRICT JSON ONLY (no prose, no markdown):
{
  "score": {
      "precision": <float>,
      "recall": <float>,
      "f1": <float>
  },
  "feedback": <str>"
}

Example feedback:
<Example Feedback>
```

The textual feedback is appended to the archive and shown to the Designer in the next iteration, enabling reflective learning.

ICD-10 Coding Tool List

The framework provides a curated library of coding tools accessible to both the Designer and Coder agents. These tools are mainly adopted from the simple-icd-10-cm resource for CM codes[1], e.g.,

- `get_parent` - returns the immediate parent of a code in the ICD-10-CM hierarchy; `prioritize_blocks` disambiguates codes that can denote either a block or a category.
- `get_children` - returns the immediate children of a code; if a code name is ambiguous (both a block and a category), setting `prioritize_blocks=True` treats it as the block (e.g., "B99" example).
- `get_ancestors` - returns all ancestor nodes

[1]A full list of tools is available at https://github.com/StefanoTrv/simple_icd_10_CM

(e.g., category to block to chapter) of the given code, with optional block prioritization to resolve ambiguity.

- `get_descendants` - returns all descendant nodes of the given code, with optional block prioritization for ambiguous names.
- `is_ancestor` - returns `True` iff `a` is an ancestor of `b`; optional flags control how to interpret ambiguous block/category codes.
- `is_descendant` - returns `True` iff `a` is a descendant of `b`; with the same ambiguity controls.
- `get_nearest_common_ancestor` - returns the nearest common ancestor of `a` and `b` (or empty string if none), with optional block/category disambiguation.
- `is_leaf` - returns `True` iff the code is a leaf (has no children) in the ICD-10-CM hierarchy; supports block prioritization.
- `code_to_text` - returns the textual description associated with a given code.
- `text_to_codes` - maps natural-language text to candidate ICD-10-CM codes (by matching code descriptions).
- `load_taxonomy` - loads the internal ICD-10-CM taxonomy data structure for lookups and hierarchies.
- `is_valid_code` - checks whether the given ICD-10-CM code exists in the taxonomy.

and we additionally create more complex LLM-based tools for both ICD-10-CM (diagnoses) and ICD-10-PCS (procedures):

- `MedicalTermExtraction` - retrieves medical terms from the note using Langextract.
- `TabularIndexSearch` - retrieves hierarchical and related codes from the tabular list.
- `DescMatch` - computes similarity between ICD descriptions and note text.
- `GuidelineValidator` - applies ICD-10 coding rules to remove invalid or conflicting codes.
- `Reconciler` - merges duplicates and resolves inter-tool inconsistencies.
- `EvidenceLinker` - extracts supporting text snippets for each predicted code.
- `Judge` - assigns per-code confidence scores and keep/drop decisions.

Beyond the predefined toolset, the Designer agent is explicitly encouraged to invent new tools using the same LLM interface (`LLMAgentBase.run_text`). When proposing new operations, the Designer must specify their role, purpose, and expected input/output signatures. This design encourages creative exploration of novel reasoning and validation strategies, allowing MedDCR to continuously expand its operational toolkit beyond the initial ICD-10 functions.

- `LLMAgentBase.run_text(prompt, role, loops)` - performs controlled text generation for tasks such as code proposal, evidence linking, or reflection. Explicit roles (e.g., "coder", "judge", "reflector") are specified, and parameters such as `loops` are kept consistent with the workflow plan.

These tools collectively enable workflow designs that emulate professional coding processes, combining retrieval, validation, and reasoning for robust code prediction.

E Potential Risks

While MedDCR is designed to improve the reliability of automated medical coding, several potential risks should be acknowledged. First, as the underlying GPT-based models are not clinically validated or fine-tuned for medical reasoning, they may generate inaccurate or incomplete workflows that lead to incorrect code assignments. Such errors could propagate in downstream analyses or billing if deployed without human oversight. Second, reflective feedback generated by large models may include unverifiable or inconsistent rationales, which, if misinterpreted, could undermine trust in automated explanations. Third, because workflow optimisation relies on data derived from de-identified clinical corpora, there remains a residual risk of bias reflecting documentation practices from specific institutions or regions. Finally, automated systems like MedDCR should not be used for direct clinical decision-making or patient care without rigorous validation and professional review. These risks highlight the importance of using MedDCR strictly for research and development under appropriate governance and human-in-the-loop supervision.