AutoHealth: Uncertainty-Aware Multi-Agent Framework for Autonomous Health Prediction

Anonymous ACL submission

Abstract

Developing effective and reliable health prediction models traditionally requires close collaboration between clinicians and machine learning experts, which is costly and limits efficiency. Recent advances in large language model (LLM) agents offer the potential for clinicians to autonomously develop prediction models through natural language interaction. However, existing agent-based systems remain limited in handling heterogeneous health data, complex prediction tasks, and the reliability requirements of healthcare applications. In this work, we present *AutoHealth*, an uncertainty-aware multi-agent framework for autonomous health prediction. Given task descriptions, *AutoHealth* dynamically constructs specialized agents that collaboratively perform data exploration, strategy design, code generation, model training, evaluation, and iterative refinement. Beyond optimizing predictive performance, the framework explicitly models uncertainty and generates comprehensive reports to support trustworthy clinical decision-making. To enable robust evaluation, we curate a real-world benchmark comprising 17 tasks across diverse health data modalities and learning settings. On this benchmark, *AutoHealth* completes all tasks and outperforms state-of-the-art baselines by 31.6% in prediction performance and 51.4% in uncertainty estimation. These results demonstrate the strong capability of LLM agents in automated health data modeling.

1 Introduction

Artificial intelligence (AI)-powered health prediction has become increasingly important in precision medicine (Purba et al., 2019; Rajpurkar et al., 2022). However, developing effective and reliable prediction models traditionally requires substantial programming expertise and close collaboration between clinicians and AI specialists, resulting in high costs and limited efficiency. Recent advances in LLMs, natural language interfaces, and agent-based AI systems are beginning to fundamentally reshape the model development process (Zhang et al., 2024; Islam et al., 2024; Wang et al., 2025). Automated data science agents (Chen et al., 2025; Hong et al., 2025; Jing et al., 2024) have demonstrated strong capabilities in reasoning, coding, experimentation, and iterative refinement. Together, these developments create the possibility for clinicians and health researchers to autonomously develop prediction models without extensive programming expertise, potentially lowering the barriers to AI-powered health modeling.

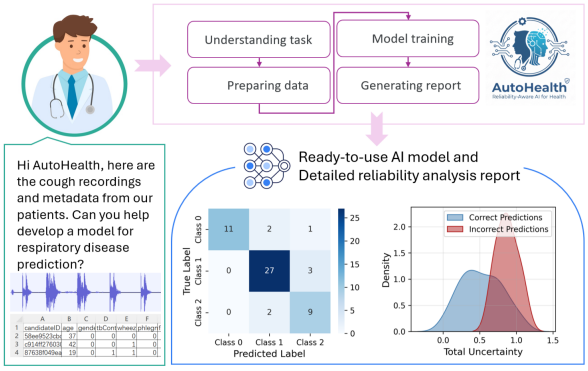


Fig. 1: *Overview of AutoHealth.* Given natural language task descriptions and raw health data, *AutoHealth* coordinates multiple specialized LLM agents to autonomously perform data exploration, modeling, training, evaluation, and uncertainty-aware reporting.

Despite this potential, existing agent-based systems remain insufficient for health data applications. General-purpose systems such as *Claude Code* often require substantial prompt engineering and technical expertise, limiting their accessibility for health researchers and clinicians (Liu et al., 2026). Moreover, their reliance on external cloud-based services may introduce considerable computational costs and raise privacy and regulatory concerns due to the need to upload sensitive health data (Tamkin et al., 2024).

Other end-to-end data science agents, such as *DS-Agent* (Guo et al., 2024) and *CodeAgent* (Zhang

et al., 2024), also exhibit fundamental limitations in healthcare settings. First, these systems provide limited support for multi-source, multimodal, and non-machine-learning-friendly health data, making it difficult to capture the heterogeneity and irregularity inherent in real-world healthcare environments. Second, they often rely on predefined pipelines or templated solutions, lacking task-conditioned reasoning mechanisms that can flexibly adapt modeling strategies to task-specific objectives and data characteristics. Third, existing agents primarily optimize predictive accuracy, whereas reliability-related aspects, such as interpretability and uncertainty quantification, are rarely treated as first-class objectives, thereby limiting their applicability to high-stakes clinical and public health decision-making (Cheng and Suo, 2025).

To this end, we propose *AutoHealth*, a LLM-agent based system for autonomous and uncertainty-aware health data modeling. As illustrated in Figure 1, users interact with *AutoHealth* through natural language descriptions of clinical objectives, and dataset characteristics, without requiring manual programming or pipeline design. Based on these instructions and raw health data, *AutoHealth* autonomously performs task understanding, data exploration, model construction, training, and evaluation through iterative multi-agent collaboration. The system produces both deployable prediction models and structured experimental reports, enabling reliable and reproducible health data modeling with minimal human intervention.

Methodologically, *AutoHealth* introduces a closed-loop multi-agent framework. To support heterogeneous healthcare environments, a dedicated data exploration agent performs code-driven analysis and preprocessing for multi-source, multimodal, and irregular health data. To overcome the limitations of predefined solution templates, a coordination agent iteratively refines modeling strategies based on task objectives and intermediate feedback. Beyond predictive accuracy, *AutoHealth* explicitly incorporates uncertainty quantification and automatically generates structured experimental reports to support trustworthy, interpretable, and risk-aware model deployment.

To rigorously evaluate autonomous health data modeling under heterogeneous data conditions and reliability constraints, we construct a challenging real-world benchmark comprising 17 health prediction tasks spanning six data modalities (tabular, image, time series, free text, audio, and graph) and six prediction settings (classification, regression, segmentation, survival analysis, forecasting, and link prediction). Extensive experiments on this benchmark demonstrate that *AutoHealth* consistently outperforms strong baselines across repeated runs, achieving a nearly 100% task success rate and improving normalized prediction and uncertainty scores over the strongest baseline by 31.6% and 51.4%, respectively.

Our contributions are summarized as follows:

- We propose *AutoHealth*, a closed-loop agentic system that autonomously constructs predictive models and reliability-aware analysis reports from natural language task descriptions and raw health data.
- We introduce a multi-agent coordination paradigm that enables data-aware planning and adaptive model construction beyond template-based automated pipelines for heterogeneous health prediction tasks.
- We construct a challenging real-world health prediction benchmark and conduct extensive evaluations, demonstrating that *AutoHealth* achieves consistent and substantial improvements over strong baselines. The benchmark and code are available from `https://anonymous.4open.science/r/AutoHealth-46E0`.

2 Background

2.1 Uncertainty-aware Machine Learning

In high-stakes domains such as healthcare, it is critical not only for models to achieve strong predictive performance but also to provide calibrated uncertainty estimates when deployed in decision support settings (Kompa et al., 2021; Begoli et al., 2019; Xia et al., 2024). Uncertainty information can help identify ambiguous or out-of-distribution inputs, enable selective prediction, and support safer human–AI collaboration. Prior work has explored a range of uncertainty quantification techniques for AI, including Bayesian neural networks (Gal and Ghahramani, 2016), deep ensembles (Lakshminarayanan et al., 2017), and test-time data augmentation (Wang et al., 2019). However, many health prediction models still lack built-in uncertainty estimation mechanisms, and evaluation protocols often focus primarily on accuracy while overlooking calibration, coverage, or abstention behavior (Van Calster et al., 2019). Overall, prior work underscores

the importance of uncertainty quantification for reliable AI-based health prediction, although it is still frequently under-emphasized in model development and evaluation.

2.2 LLM-based Data Science Agents

Recent LLMs, such as *Claude Sonnet 4* (Anthropic, 2025b) and *GPT-5* (OpenAI, 2025), have demonstrated strong capabilities in code generation and program assistance, enabling specialized coding agents such as *Claude Code* (Anthropic, 2025a). These systems substantially reduce programming effort, but applying them to health data modeling still often requires prompt engineering, environment setup, and methodological expertise, limiting accessibility for non-technical health researchers.

Early work in *automated data science* mainly focused on single-agent systems that translate natural language instructions into executable pipelines. Representative frameworks such as *AutoML-GPT* (Zhang et al., 2023), *HuggingGPT* (Shen et al., 2023), and *Prompt2Model* (Viswanathan et al., 2023) established end-to-end workflows integrating preprocessing, model selection, and evaluation. However, single-agent approaches are constrained by limited context and reasoning capacity, making them less effective for complex real-world health data.

Recent studies therefore increasingly adopt *multi-agent LLM systems*, where specialized agents collaboratively solve complex tasks. *DS-Agent* (Guo et al., 2024) introduces case-based retrieval to reuse prior modeling experiences, while *Data-Interpreter* (Hong et al., 2025) incorporates trajectory-level feedback for iterative refinement. *AutoML-Agent* (Trirat et al., 2025) further combines retrieval and feedback within a unified framework, but primarily targets benchmark-style datasets and conventional optimization objectives, with limited support for heterogeneous health data and uncertainty-aware modeling.

Several frameworks also explore *domain-specific* automation. *AutoMMLab* (Yang et al., 2025) focuses on computer vision pipelines, and *AutoKaggle* (Li et al., 2024) targets competition-style workflows. *OpenLens AI* (Cheng and Suo, 2025) is most relevant to healthcare applications, emphasizing exploratory health informatics analysis and visual reporting. However, these systems generally provide limited support for systematic health prediction modeling and rarely incorporate explicit uncertainty quantification for reliable decision-making.

Overall, existing systems typically address isolated aspects of automated modeling, such as pipeline reuse or feedback refinement, but rarely integrate heterogeneous data exploration, task-conditioned model construction, and uncertainty-aware evaluation within a unified framework.

3 AutoHealth

We now introduce the specific design of *AutoHealth*. The framework consists of five agents and an iterative workflow, as illustrated in Figure 2. An overview of the system and the specifications of different stages are presented below.

3.1 System Overview

The five agents in *AutoHealth* play different roles and functions:

- **Meta-Agent** ($\mathcal{A}_m$) serves as the central coordinator and user interface. It translates natural language modeling objectives into executable workflows and manages iterative multi-agent collaboration.
- **Data-Agent** ($\mathcal{A}_d$) performs health data exploration, preprocessing, and quality assessment for heterogeneous and multi-source datasets.
- **Design-Agent** ($\mathcal{A}_p$) generates task-specific modeling strategies, including model selection, feature design, and training configurations.
- **Coding-Agent** ($\mathcal{A}_c$) converts experimental plans into executable AI pipelines and iteratively updates implementations during optimization.
- **Report-Agent** ($\mathcal{A}_r$) summarizes experimental results, uncertainty analyses, and model limitations into structured and interpretable reports.

Under the coordination of the Meta-Agent $\mathcal{A}_m$, *AutoHealth* follows an iterative closed-loop workflow consisting of three stages: preparation, planning, and execution. At each iteration $i < N$, the workflow progressively refines the modeling solution based on intermediate results and feedback:

- **Preparation Stage.** The Data-Agent $\mathcal{A}_d$ performs data exploration and quality assessment, producing a structured data profile R_i^d.
- **Planning Stage.** Conditioned on the task specification, data profile R_i^d, and previous feedback $\{M_1, M_2, \cdots, M_{i-1}\}$, the Design-Agent $\mathcal{A}_p$ generates an experimental plan R_i^p, including model architectures, training strategies, and evaluation protocols.
- **Execution Stage.** Given R_i^d and R_i^p, the

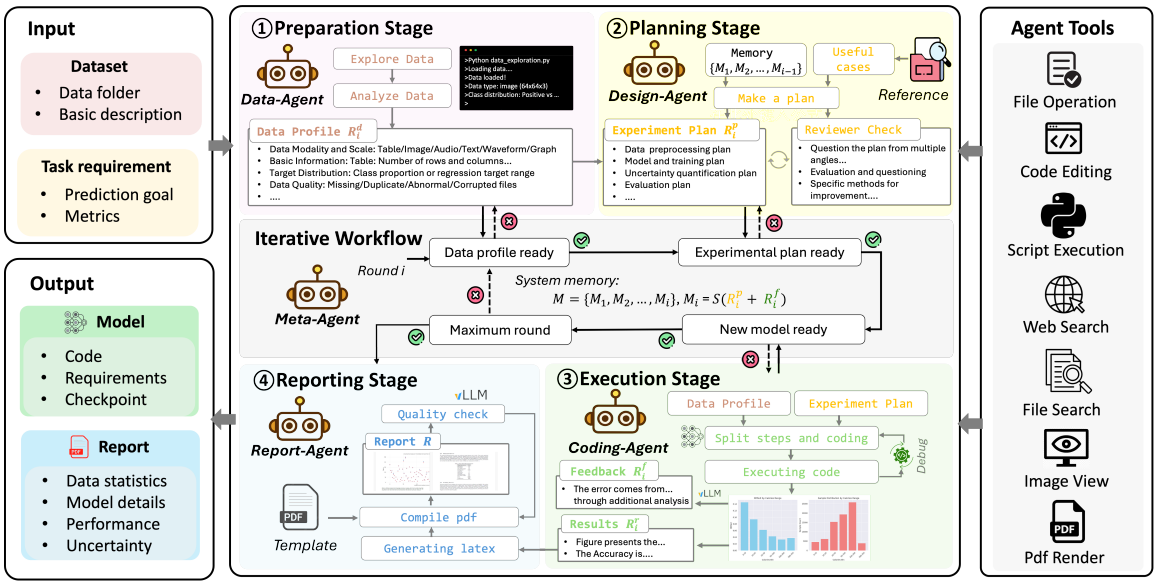


Fig. 2: *AutoHealth framework.* The system adopts a closed-loop, multi-agent architecture coordinated by a *Meta-Agent*, which orchestrates an iterative workflow spanning preparation, planning, execution, and reporting. Specialized agents operate at each stage to enable modular, reliable, and reproducible health data modeling.

Coding-Agent $\mathcal{A}_c$ implements the pipeline, trains models, evaluates performance, and produces an experimental record R_i^r together with feedback R_i^f for subsequent refinement.

Final Reporting Stage. After the final iteration N, the Report-Agent consolidates R_N^r into a uncertainty-aware, application-oriented report R, which serves as the final system output.

3.2 Preparation Stage

Given raw health data and task specifications, the Data-Agent $\mathcal{A}_d$ produces a structured data profile R_i^d for downstream modeling with the following strategies:

Code-driven Data Understanding. Rather than relying on manual dataset descriptions, $\mathcal{A}_d$ directly analyzes raw data through executable code, including file inspection, loading, and programmatic exploration.

Structured Exploration Across Modalities. Standardized exploration objectives are defined for seven common health data types, including tabular, time-series, image, segmentation, audio, text, and graph data. The analysis covers data structure, quality, and feature–target relationships.

Data-informed Modeling Guidance. Based on the exploration results, $\mathcal{A}_d$ generates preprocessing recommendations and reference code to support downstream modeling decisions.

3.3 Planning Stage

Given the data profile R_i^d, task specification, and memory from previous rounds $\{M_1, \ldots, M_{i-1}\}$, the Design-Agent $\mathcal{A}_p$ generates an experimental plan R_i^p with the following steps:

Retrieval-augmented Planning. $\mathcal{A}_p$ combines local case retrieval and web search tools to ground planning decisions in validated modeling practices. To support **uncertainty-aware modeling**, we further construct a structured *uncertainty estimation knowledge base* that maintains method-level entries describing the estimation type, applicable tasks, usage strategy, advantages, limitations, computational cost, and recommendation level of different uncertainty estimation approaches. For example, Deep Ensemble is annotated as highly reliable for high-risk tasks but computationally expensive, whereas Temperature Scaling is recommended for probability calibration with low implementation complexity but limited capability for out-of-distribution detection. Given a specific task, $\mathcal{A}_p$ retrieves and ranks candidate uncertainty estimation strategies according to task objectives, data characteristics, and reliability requirements, enabling context-aware selection of uncertainty-aware modeling pipelines.

Feedback-aware Refinement. The system maintains a memory $\mathcal{M} = \{M_1, \ldots, M_{i-1}\}$ containing previous plans and execution feedback. During planning, $\mathcal{A}_p$ adaptively revises modeling strategies based on historical performance and cur-

rent task conditions.

Iterative Plan Review. An auxiliary LLM-based reviewer critiques generated plans before execution, enabling iterative refinement of R_i^p.

3.4 Execution Stage

Given R_i^d and R_i^p, the Coding-Agent $\mathcal{A}_c$ implements the experimental pipeline and produces an execution record R_i^r together with feedback R_i^f. Key designs include:

Step-wise Execution. Instead of generating a monolithic pipeline, $\mathcal{A}_c$ decomposes tasks into sequential executable steps, improving robustness and debuggability.

Performance and Uncertainty Evaluation. Beyond predictive metrics, $\mathcal{A}_c$ systematically evaluates calibration, uncertainty behavior, and potential failure modes, while logging visual diagnostics and experimental statistics.

VLLM-assisted Reflective Analysis. A Vision Large Language Model (VLLM)-based analysis tool examines execution results, visualizations, and logs to identify issues such as overfitting or data leakage, producing structured feedback R_i^f. After each iteration, the Meta-Agent $\mathcal{A}_m$ summarizes R_i^p and R_i^f into a compact memory unit M_i, enabling cumulative cross-round optimization.

3.5 Reporting Stage

After the final iteration N, the Report-Agent $\mathcal{A}_r$ summarizes R_N^r into a structured report R_N. The following principles are used:

Schema-driven Reporting. Reports follow a standardized schema covering data analysis, model performance, and uncertainty assessment.

Application-oriented Interpretation. The report emphasizes practical guidance, including reliability-aware interpretation and recommendations for human review under uncertain predictions.

Visual Quality Review. A VLLM-based reviewer checks the consistency and clarity of figures, tables, and report text before finalization. The report along with the model serves as the final output of the system.

4 Experiments and Results

4.1 Experimental Setup

Benchmark. To quantitatively evaluate *Auto-Health*, we curate a benchmark comprising 17 tasks (Table 1). The datasets are drawn from heterogeneous sources, and task-specific evaluation metrics are adopted, such as accuracy for classification and root mean squared logarithmic error (RMSLE) for regression. Additional details on the datasets and evaluation metrics are provided in Appendix A.1. Compared with prior studies (Guo et al., 2024; Trirat et al., 2025), our benchmark covers a broader range of task types, posing a more stringent test of system generalizability.

Evaluation Metrics. Following prior work (Trirat et al., 2025), we evaluate the system using success rate (SR) and normalized performance score (NPS), and report a comprehensive score (CS) defined as $\text{CS} = 0.5 \times \text{SR} + 0.5 \times \text{NPS}$.

Specifically, SR measures task completeness with two components: successful model training and uncertainty quantification. Completing both yields $\text{SR} = 1$, completing only one yields $\text{SR}= 0.5$, and completing neither yields $\text{SR}= 0$. NPS evaluates overall model quality by jointly considering task performance (namely *Pred. Score*) and uncertainty behavior (*Unc. Score*). For accuracy-based metrics, raw scores are used directly. For loss-based metrics (termed as s), both performance and uncertainty metrics are transformed as $S = 0.1/(0.1 + s)$ to ensure scale consistency. The transformed performance and uncertainty scores are then equally weighted to compute NPS.

Baselines. We compare our system with three categories of baselines. (1) *naïve LLM*: an LLM is prompted with the same task specification, and the generated code is executed manually to record performance. (2) *General coding agents*: we use *Claude Code* with comparable prompts to ensure a fair comparison. Prompts used for *LLM* and *Claude Code* can be found in Appendix A.2. (3) *Advanced data science agents*: we evaluate representative systems including *DS-Agent* (Guo et al., 2024) and *AutoML-Agent* (Trirat et al., 2025).

Implementation Details. Unless otherwise specified, all agents are implemented using *DeepSeek-V3.2* as the backbone model. When VLLM is needed, we use *GLM-4.6V*. For all agent systems, including *DS-Agent* and *AutoML-Agent*, we set the maximum number of iteration rounds to $N = 5$. Each method is run independently three times, and we report the mean and standard deviation over each metric unless otherwise specified. All experiments are conducted on an Ubuntu server with four NVIDIA A100 GPUs. To ensure fair comparison, we use the same runtime environment as *DS-Agent*, with all required libraries pre-installed via the provided skeleton scripts.

Table 1: Benchmark tasks, modalities and major evaluation metrics.

Modality	Task Type	ID	Basic Task Description	Metric
Tabular	Classification	1	predict whether a person is a smoker based on clinical measurements and basic health indicators	Accuracy
		2	predict in-hospital mortality for admitted patients	Accuracy
		3	predict the probability of each mechanism-of-action (MoA) label for a treated sample	Log Loss
	Regression	4	predict how many calories were burned during a workout	RMSLE
		5	predict post-transplant event-free survival risk for allogeneic HCT patients	C-index
Image	Classification	6	classify each skin lesion image as benign or malignant	F1
		7	classify each skin lesion image as benign or malignant (melanoma-related).	Accuracy
	Segmentation	8	segment retinal blood vessels from a fundus image	Dice
Time series	Classification	9	classify whether an ECG recording indicates a healthy status	F1
		10	predict whether a sensor sequence is a target gesture and the gesture class	Average F1
		11	classify harmful brain activity patterns from EEG spectrograms	KL divergence
	Forecasting	12	predict future COVID-19 case counts for multiple countries using historical observations	MAE
Free text	Classification	13	predict whether the described behavior patterns indicate ASD-related signs	Accuracy
		14	predict the sentiment category of a COVID-19 related tweet	Accuracy
Audio	Classification	15	predict the respiratory disease class for each candidate based on metadata and audio signals	Macro-F1
		16	detect whether a speaker exhibits dysarthric speech patterns from an audio recording	Accuracy
Graph	Link prediction	17	predict whether a pair of drugs has an interaction edge in the graph	Hits@20

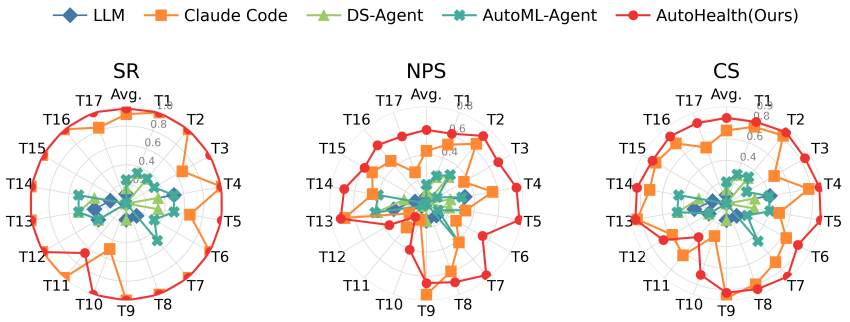


Fig. 3: Task-level performance comparison. Avg presents the averaged score across all 17 tasks.

4.2 Main Results

Figure 3 summarizes the task-level comparison across SR, NPS, and CS, while Table 2 further presents the performance of prediction and uncertainty estimation, respectively. Detailed task-level metrics are provided in Appendix B.

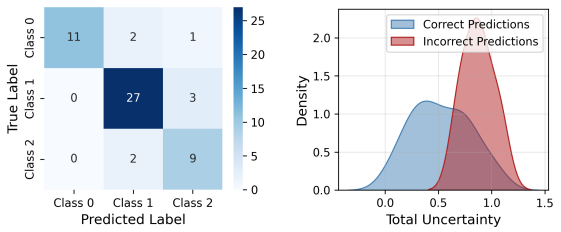
Success Rate. *AutoHealth* completes nearly all tasks in heterogeneous health data modalities and learning settings, producing an average SR of 0.980, while the strongest baseline, *Claude Code*, reaches 0.922. In contrast, general automated data-science agents such as *DS-Agent* and *AutoML-Agent* fail on many tasks because their generated code often cannot properly adapt to heterogeneous health data and may break under data mismatches without extensive iterative debugging. These findings suggest that existing automated pipelines remain brittle when faced with the diversity and irregularity of real-world health data.

Normalized Performance Score. Among successfully completed tasks, *AutoHealth* also achieves stronger predictive and uncertainty-aware performance. Compared with *Claude Code*, *AutoHealth* improves the average prediction score from 0.512 to 0.674 (31.6% relative improvement) and the average uncertainty estimation score from 0.358 to 0.542 (51.4% relative improvement). This gain is important because the benchmark evaluates not only whether an agent can produce a runnable model, but also whether the resulting model provides reliable uncertainty estimates for health-related decision making.

Table 2: Detailed performance comparison. Prediction and uncertainty scores are reported as mean ± standard deviation over three runs.

Method	SR	Pred. Score	Unc. Score	NPS	CS
LLM	0.098	0.099±0.125	0.020±0.034	0.059	0.079
Claude Code	0.922	0.512±0.075	0.358±0.119	0.435	0.678
DS-Agent	0.157	0.200±0.148	0.000±0.000	0.100	0.128
AutoML-Agent	0.245	0.313±0.095	0.000±0.000	0.157	0.201
AutoHealth	**0.980**	**0.674±0.083**	**0.542±0.106**	**0.608**	**0.794**

Comprehensive Score. The same trend is reflected in the comprehensive score. *AutoHealth* obtains the highest CS of 0.794, compared with 0.678 for *Claude Code* and much lower scores for



(a) Confusion matrix (b) Uncertainty distribution

Fig. 4: Excerpt of the generated report for Task 15 respiratory disease prediction. Full report is given in Appendix C.2.

the other automated agents. The average CS shows that this advantage is consistent at the aggregate level, while the task-level results show that the improvement is consist across tasks.

4.3 Report Quality Assessment

Case Study. To illustrate the report generation capability of *AutoHealth*, we present a representative case study based on Task 15, a multi-modal respiratory disease prediction task involving structured clinical variables and respiratory audio recordings. Due to its multi-modal nature and severe class imbalance, this task serves as a representative real-world healthcare scenario. The complete automatically generated report is provided in Appendix C.2, with selected results shown in Figure 4.

Given the task requirement, *AutoHealth* built a dual-branch late-fusion neural architecture for integrating tabular clinical features and audio embeddings, and adapt a deep ensemble strategy to improve robustness and uncertainty estimation. The generated report includes standard predictive evaluation, calibration analysis, and uncertainty assessment. As shown in Figure 4a, the model achieves balanced multi-class prediction performance despite severe class imbalance. Figure 4b shows that prediction errors are associated with higher uncertainty, supporting confidence-aware decision-making and risk-aware clinical deployment.

LLM-as-a-judge Assessment. To evaluate the quality and clinical usefulness of the automatically generated reports, we further designed an LLM-based evaluation framework from the perspective of clinical AI deployment. Specifically, GPT-5 was prompted to assess each report across multiple dimensions, including clinical readability, reliability and calibration awareness, deployment readiness, scientific rigor, and overall clinical usefulness. Each sub-dimension was scored on a 1-5 scale with standardized evaluation criteria. Details are introduced in Appendix C.1. We then aggregated the fine-grained scores into several high-level metrics and summarized their distributions across tasks, as shown in Table 3.

Table 3: LLM-based report evaluation metrics.

Metric (Scale: 1-5)	Mean	Std	Min	Median	Max
Clinical Readability	4.21	0.49	3.50	4.00	4.75
Clinical Reliability	3.97	0.73	2.25	4.25	4.75
Deployment Readiness	3.78	0.78	2.33	4.00	5.00
Scientific Rigor	4.67	0.24	4.33	4.67	5.00
Overall Assessment	4.33	0.49	3.67	4.33	5.00
Overall Score	4.19	0.48	3.28	4.25	4.83

Overall, the automatically generated reports achieved strong evaluation results across tasks, with an average overall score of 4.19/5. The reports showed particularly high scientific rigor (mean: 4.67) and strong clinical readability (mean: 4.21), suggesting that *AutoHealth* can generate clinically interpretable and methodologically transparent reports while appropriately discussing uncertainty, deployment risks, and model limitations. Importantly, the noticeable variance across tasks suggests that the LLM-based evaluator could meaningfully differentiate report quality rather than assigning uniformly positive scores.

4.4 Additional Analysis

Ablation Study. We first examine the contributions of data exploration and step-wise plan decomposition and execution to successfully completing all tasks. We hypothesize that these design components are key to generalizing across heterogeneous data modalities and learning settings. We compare the SR of the full AutoHealth system with two degraded variants: *AutoHealth-S* (without step decomposition) and *AutoHealth-SD* (without both step decomposition and data exploration). The results are summarized in Table 4. Removing step decomposition already results in a notable reduction in success rate, while further disabling data exploration leads to an even more pronounced performance drop. This trend underscores the complementary importance of structured planning and data-aware reasoning in *AutoHealth*.

Table 4: Ablation results on SR.

Metric	AutoHealth	AutoHealth-S	AutoHealth-SD
SR (%)	98.0	70.6	41.2

We then examine whether uncertainty estimation is supported by the designed workflow rather than by generic prompting. We ablate uncertainty quantification templates during planning (*AutoHealth-T*) and VLLM-assisted reflective analysis during

execution (*AutoHealth-R*) across both tabular and non-tabular tasks. As shown in Table 5, the full *AutoHealth* achieves the highest average normalized uncertainty score across all evaluated tasks. Removing uncertainty templates reduces the score from 0.614 to 0.542, while removing reflective analysis further reduces it to 0.470. The consistent degradation across task types indicates that reliable uncertainty estimation depends on both reusable uncertainty-aware planning and adaptive feedback-driven reflection.

Table 5: Uncertainty ablation results on Unc. Score.

Metric	Claude Code	AutoHealth	AutoHealth-T	AutoHealth-R
Unc. Score	0.408	0.614	0.542	0.470

Multi-agent Attribution. To isolate the contribution of the multi-agent design, we construct *AutoHealth-SingleAgent*, a matched single-agent variant with the same backbone, tools, search space, and refinement budget as the full system. The only removed components are multi-agent decomposition, role specialization, and closed-loop inter-agent coordination. Since the single-agent variant is evaluated on all benchmark tasks, Table 6 reports the full-task aggregate rather than a representative-task subset, with task-level details provided in Appendix Table 20. The single-agent variant underperforms the full *AutoHealth* across task completion, normalized prediction, and uncertainty-aware modeling. This indicates that the gains are not solely due to access to tools or repeated refinement, but also depend on structured multi-agent coordination.

Table 6: Multi-agent attribution across all tasks.

Method	SR	Pred.	Unc.	NPS	CS
Single-agent	0.765	0.459±0.019	0.297±0.157	0.378	0.571
AutoHealth	**0.980**	**0.674±0.083**	**0.542±0.106**	**0.608**	**0.794**

Backbone Robustness. To examine whether the gains depend on a specific LLM backbone, we further evaluate five representative tasks using *DeepSeek-V3.2*, *DeepSeek-V3*, *Gemini-3-flash*, *Gemini-2.0-flash*, and *Qwen3.5-27B*. As summarized in Table 7, *AutoHealth* consistently achieves higher prediction and uncertainty scores than *Claude Code* across different backbones. This suggests that the improvements are not simply due to using a stronger language model, but are associated with the structured agent workflow.

Cost and Efficiency Analysis. Table 8 presents the token usage, monetary cost, and wall-clock time for selected tasks (full results can be found from Appendix Table 9, and the execution-time proportion of each stage is summarized in Appendix Figure 5). Across the 17 tasks, the total token consumption remains within a moderate range, while the monetary cost is consistently low, typically below 6 RMB per task. Importantly, *AutoHealth* demonstrates fast iterative cycles, with most tasks completing within 2–6 hours (5 runs), even for tasks involving gigabyte-scale data. This efficiency is largely attributed to effective and structured multi-agent coordination, which substantially reduces redundant token consumption across iterations. By integrating task understanding, model construction, execution, and feedback into a unified end-to-end agentic workflow, *AutoHealth* minimizes human intervention and iteration overhead. Together, these results indicate that *AutoHealth* enables rapid and affordable end-to-end experimentation, thereby lowering the barrier to large-scale empirical studies and accelerating scientific discovery in real-world health data modeling.

Table 7: Backbone robustness on five representative tasks (T4, T5, T8, T10, and T15).

Backbone	Pred. Score		Unc. Score	
	AutoHealth	Claude Code	AutoHealth	Claude Code
DeepSeek-V3.2	**0.643±0.069**	0.592±0.094	**0.636±0.067**	0.382±0.018
DeepSeek-V3	**0.578±0.086**	0.523±0.246	**0.557±0.100**	0.447±0.077
Gemini-3-flash	**0.675±0.027**	0.654±0.062	**0.627±0.073**	0.409±0.073
Gemini-2.0-flash	**0.480±0.207**	0.187±0.094	**0.225±0.059**	0.000±0.000
Qwen3.5-27B	**0.640±0.027**	0.613±0.019	**0.600±0.016**	0.548±0.024

Table 8: Token and time cost analysis. Input, output and cache token counts are reported in millions (M).

Task	**Data size**	**Input**	**Output**	**Cache**	**Cost (RMB)**	**Time (min)**
T1	11.03MB	1.58	0.20	0.93	2.1	119.8
T5	9.87MB	2.65	0.38	1.49	3.8	246.1
T6	143.5MB	3.49	0.55	1.76	5.5	478.7
T9	7.75MB	1.81	0.22	1.14	2.2	123.2
T11	6.6GB	3.73	0.47	1.83	5.6	634.1
T15	1.9GB	3.03	0.38	1.69	4.2	187.5
T17	102MB	3.38	0.47	1.77	5.0	419.5

5 Conclusions

In this work, we introduced *AutoHealth*, an uncertainty-aware multi-agent framework for autonomous health data modeling. Experimental results on diverse real-world tasks show that *AutoHealth* consistently outperforms existing general-purpose coding and data-science agents in both task completion and modeling reliability. We believe that this work contributes to the development of trustworthy LLM agent-based systems for healthcare and may inspire further exploration of their limitations and future potential.

Limitations

One limitation of *AutoHealth* is that the framework currently optimizes each prediction task independently. Although the system can iteratively improve model design and coding strategies within a task, the accumulated experience and domain-specific skills are not explicitly stored, generalized, or transferred across tasks. As a result, the framework does not yet possess a persistent mechanism for continual learning or reusable medical modeling knowledge. Future work could explore long-term memory modules, skill libraries, or cross-task agent adaptation to enable cumulative learning and more efficient autonomous modeling over time.

In addition, although our benchmark contains 17 diverse tasks spanning multiple health data modalities and learning settings, all tasks are derived from publicly available competitions and benchmark datasets. Such datasets may differ substantially from real-world clinical environments in terms of data quality, distribution shifts, missingness patterns, workflow constraints, and regulatory requirements. Therefore, the current evaluation may not fully capture the complexity and operational challenges of deploying autonomous AI agents in real clinical settings. Future studies should further validate *AutoHealth* using large-scale real-world clinical datasets and prospective healthcare scenarios.

Ethical Considerations and Potential Risks

This paper presents a methodological contribution aimed at advancing autonomous data science and machine learning systems for health data modeling. The proposed framework is designed to improve reliability, transparency, and uncertainty awareness in predictive modeling, which are widely recognized as important considerations for healthcare applications. We do not foresee any immediate negative societal impacts beyond those commonly associated with machine learning research in healthcare.

All datasets used in this study are publicly available benchmark datasets or competition datasets that have been previously released for research purposes. The data are anonymized and do not contain personally identifiable information. Therefore, this work does not involve direct interaction with human participants, access to private clinical records, or collection of sensitive personal data, and no additional ethical approval was required for this study.

References

Anthropic. 2025a. Claude Code. claude.com/product/claude-code.

Anthropic. 2025b. System card: Claude Opus 4 & Claude Sonnet 4. anthropic.com/claude-4-system-card.

Edmon Begoli, Tanmoy Bhattacharya, and Dimitri Kusnezov. 2019. The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence*, 1:20–23.

Ke Chen, Peiran Wang, Yaoning Yu, Xianyang Zhan, and Haohan Wang. 2025. Large language model-based data science agent: A survey. *arXiv preprint arXiv:2508.02744*.

Yuxiao Cheng and Jinli Suo. 2025. OpenLens AI: Fully autonomous research agent for health infomatics. *arXiv preprint arXiv:2509.14778*.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059. PMLR.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024. DS-agent: Automated data science by empowering large language models with case-based reasoning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 16813–16848. PMLR.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, and 9 others. 2025. Data interpreter: An LLM agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19796–19821, Vienna, Austria. Association for Computational Linguistics.

Md. Ashraful Islam, Mohammed Eunus Ali, and Md. Rizwan Parvez. 2024. MapCoder: Multi-agent code generation for competitive problem solving. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4912–4944, Bangkok, Thailand. Association for Computational Linguistics.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2024. DSBench: How far are data science agents from becoming data science experts? *arXiv preprint arXiv:2409.07703*.

Benjamin Kompa, Jasper Snoek, and Andrew L. Beam. 2021. Second opinion needed: Communicating uncertainty in medical machine learning. *npj Digital Medicine*, 4(4).

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30, pages 6402–6413.

Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, Wanjun Zhong, Wangchunshu Zhou, Wenhao Huang, and Ge Zhang. 2024. AutoKaggle: A multi-agent framework for autonomous data science competitions. *arXiv preprint arXiv:2410.20424*.

Jiacheng Liu, Xiaohan Zhao, Xinyi Shang, and Zhiqiang Shen. 2026. Dive into claude code: The design space of today's and future ai agent systems. *arXiv preprint arXiv:2604.14228*.

OpenAI. 2025. GPT-5 system card. openai.com/index/gpt-5-system-card.

Jan Horas Veryady Purba, Muhamad Ratodi, Mumuh Mulyana, Soegeng Wahyoedi, Ririn Andriana, K. Shankar, and Phong Thanh Nguyen. 2019. Prediction model in medical science and health care. *International Journal of Engineering and Advanced Technology*, 8(6S3):815–818.

Pranav Rajpurkar, Emma Chen, Oishi Banerjee, and Eric J. Topol. 2022. AI in health and medicine. *Nature Medicine*, 28(1):31–38.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. In *Advances in Neural Information Processing Systems*, volume 36, pages 38154–38180.

Alex Tamkin, Miles McCain, Kunal Handa, Esin Durmus, Liane Lovitt, Ankur Rathi, Saffron Huang, Alfred Mountfield, Jerry Hong, Stuart Ritchie, Michael Stern, Brian Clarke, Landon Goldberg, Theodore R. Sumers, Jared Mueller, William McEachen, Wes Mitchell, Shan Carter, Jack Clark, and 2 others. 2024. Clio: Privacy-preserving insights into real-world AI use. *arXiv preprint arXiv:2412.13678*.

Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. 2025. AutoML-Agent: A multi-agent LLM framework for full-pipeline AutoML. In *Proceedings of the 42nd International Conference on Machine Learning*. Poster session.

Ben Van Calster, David J. McLernon, Maarten van Smeden, Laure Wynants, and Ewout W. Steyerberg. 2019. Calibration: The achilles heel of predictive analytics. *BMC Medicine*, 17(230). On behalf of Topic Group "Evaluating diagnostic tests and prediction models" of the STRATOS initiative.

Vijay Viswanathan, Chenyang Zhao, Amanda Bertsch, Tongshuang Wu, and Graham Neubig. 2023. Prompt2Model: Generating deployable models from natural language instructions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 413–421, Singapore. Association for Computational Linguistics.

Guotai Wang, Wenqi Li, Michael Aertsen, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. 2019. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks. *Neurocomputing*, 338:34–45.

Peiran Wang, Xiaogeng Liu, and Chaowei Xiao. 2025. CVE-bench: Benchmarking LLM-based software engineering agent's ability to repair real-world CVE vulnerabilities. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4207–4224, Albuquerque, New Mexico. Association for Computational Linguistics.

Tong Xia, Ting Dang, Jing Han, Lorena Qendro, and Cecilia Mascolo. 2024. Uncertainty-aware health diagnostics via class-balanced evidential deep learning. *IEEE Journal of Biomedical and Health Informatics*, 28(11):6417–6428.

Zekang Yang, Wang Zeng, Sheng Jin, Chen Qian, Ping Luo, and Wentao Liu. 2025. AutoMMLab: Automatically generating deployable models from language instructions for computer vision tasks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(21):22056–22064.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024. CodeAgent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13643–13658, Bangkok, Thailand. Association for Computational Linguistics.

Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. 2023. AutoML-GPT: Automatic machine learning with GPT. *arXiv preprint arXiv:2305.02499*.

Appendix

The appendix of this paper is organized as follows:

- Appendix A provides additional details on the benchmark.
- Appendix B presents more detailed results.
- Appendix C presents the evaluation for generated reports.
- Appendix D specifies the design of the specialized agents in *AutoHealth*.

A Additional Introduction for Benchmark and Experiments

A.1 Benchmark

Tabular data based tasks include:

- **T1 Smoker status prediction.** This task is built on a structured health-behavior dataset where each row represents an individual's physiological signals and lifestyle attributes. The target is a binary label indicating smoking status. The goal is to learn patterns in biomarkers—such as lung capacity, cholesterol levels, blood pressure, or other body-signal indicators—that reliably distinguish smokers from non-smokers. For ML, the dataset offers opportunities to explore feature importance, uncover latent health correlations, evaluate model fairness across subgroups, and build predictive systems that highlight which physiological signals are most informative for identifying smoking-related risk profiles. More details can be found from `https://www.kaggle.com/datasets/kukuroo3/body-signal-of-smoking/data`.

- **T2 Patient survival prediction.** This task centers on a structured clinical dataset where each record captures a patient's demographic attributes, laboratory measurements, and disease-related indicators, and the target is a categorical survival outcome reflecting different stages of prognosis. The aim is to model how patterns in these medical features relate to survival trajectories, enabling ML systems to learn which biomarkers and clinical signals are most predictive of patient risk and long-term outcomes. More details can be found from `https://www.kaggle.com/datasets/joebeachcapital/cirrhosis-patient-survival-prediction/code?datasetId=3873965&sortBy=voteCount`.

- **T3 Drug response mechanism prediction.** This task uses a high-dimensional biological screening dataset where each sample corresponds to a compound tested under specific cellular conditions, and the targets are multi-label mechanism-of-action indicators describing which biological pathways the compound perturbs. The goal is to learn predictive mappings from gene-expression and cell-viability features to these MoA labels, enabling models to uncover functional signatures, distinguish subtle pathway effects, and support downstream drug-discovery insights. More details can be found from `https://www.kaggle.com/competitions/lish-moa/data?select=train_targets_nonscored.csv`.

- **T4 Calorie estimation.** This task provides a structured regression task built on *train.csv*, where each row represents a physical-activity record with features such as duration, heart-rate signals, and movement-related metrics, and the target is a continuous Calories value that quantifies energy expenditure . The goal is to learn how patterns in these activity features map to calorie burn, enabling models to capture physiological relationships, improve prediction accuracy, and support insights into how different forms of movement translate into metabolic output. More details can be found from `https://www.kaggle.com/competitions/international-medical-ai-and-data-science-calorie/data`.

- **T5 Post-HCT survival analysis.** This task is built on a synthetic but clinically grounded dataset where each row represents an allogeneic HCT patient with demographic variables, disease characteristics, transplant details, and post-treatment clinical signals, and the target is a real-valued risk score reflecting predicted survival likelihood after transplant. The challenge is to model survival outcomes in a way that is both accurate and equitable across racial groups, learning patterns in the medical features that drive post-HCT prognosis while minimizing disparities in predictive performance. More details can be found from https://www.kaggle.com/competitions/equity-post-HCT-survival-predictions/overview.

Clinical image data based tasks include:

- **T6 Skin cancer identification .** This task provides a skin-lesion image dataset where part of the training set is labeled and a substantial portion is unlabeled, and the labeled subset is heavily imbalanced with far more benign than malignant cases. The task is to build an image-classification model that predicts a probability for the LABEL variable (malignant vs. benign) for each test image, learning effectively from limited labeled data while leveraging the large unlabeled pool and mitigating class imbalance to achieve strong F1 performance. More details can be found from https://www.kaggle.com/competitions/dsm-2024-competition/overview.

- **T7 Melanoma detection.** This task is built around a curated skin-lesion image dataset in which each sample is a dermoscopic photograph paired with a binary target label indicating whether the lesion is benign or malignant. The goal is to train a classifier that learns visual patterns—such as texture, color variation, and structural irregularities—that distinguish malignant melanoma from non-cancerous lesions, enabling accurate probability predictions for the target label on unseen images. More details can be found from https://www.kaggle.com/datasets/bhaveshmittal/melanoma-cancer-dataset.

- **T8 Retinal blood vessel segmentation.** This task gives a retinal fundus image dataset in which each sample is a high-resolution photograph of the eye, paired with a pixel-level segmentation mask marking blood-vessel locations as 1 and background as 0. The goal is to train a model that learns the visual structure of retinal vasculature—capturing vessel width, branching patterns, and subtle pathological variations—to accurately predict a full-resolution vessel mask for each input image, supporting downstream diagnostic and ophthalmic analysis. More details can be found from https://www.kaggle.com/datasets/abdallahwagih/retina-blood-vessel/data.

Health time series data based tasks include:

- **T9 ECG-based arrhythmia detection.** This task uses a structured ECG time-series dataset where each row corresponds to a single patient and contains a heartbeat of ECG signal value along with metadata such as an ID and rhythm-class indicators. The target is the healthy status label, a binary outcome indicating whether the ECG is assessed as Normal or Abnormal, derived from clinical annotations in the original PhysioNet source. The goal is to learn discriminative patterns in the raw ECG waveform—morphology, rhythm regularity, and subtle signal deviations—to accurately classify cardiac health status for new, unseen patients. More details can be found from https://www.kaggle.com/competitions/ecg-ai-fast-checkup/overview.

- **T10 IMU-based gesture prediction.** This task provides a multichannel time-series sensor dataset collected from a wrist-worn device, where each sequence contains synchronized IMU motion signals along with temperature and proximity readings. The target is a gesture label indicating whether the movement corresponds to a body-focused repetitive behavior (BFRB) or a non-BFRB action, with further granularity for specific BFRB gesture types. The task is to train a model that learns discriminative temporal patterns across sensors to reliably classify these behaviors, supporting more accurate and clinically meaningful detection of BFRB-related movements. Binary-F1 on whether the gesture is one of the target or non-target types, and Macro-F1 on gesture, where all non-target

sequences are collapsed into a single non-target class, are calculated. The final score is the average of the binary F1 and the macro F1 scores. More details can be found from https://www.kaggle.com/competitions/cmi-detect-behavior-with-sensor-data.

- **T11 EEG-based seizure prediction.** This task provides a large-scale EEG dataset where each training example corresponds to a short segment of multichannel brain-wave activity recorded from critically ill patients, paired with expert-annotated vote counts for six possible harmful brain-activity patterns: seizure, LPD, GPD, LRDA, GRDA, and other. The task is to learn a model that maps raw EEG signals and derived features to a probability distribution over these six target classes, capturing subtle temporal–spectral signatures so the system can reliably classify harmful patterns and support faster, more consistent neurocritical-care assessment More details can be found from https://www.kaggle.com/competitions/hms-harmful-brain-activity-classification/overview.

- **T12 COVID-19 infection case forecasting.** This dataset provides a global COVID-19 time-series table where each row records a specific country or province on a given date, along with key epidemiological counts: Confirmed, Recovered, and Deaths, supported by geographic metadata such as latitude and longitude. The task is to model how these case trajectories evolve over time—treating the target as any of these outcome variables depending on the analysis goal—so that ML systems can capture temporal patterns, forecast future case numbers, and uncover relationships between regional characteristics and pandemic dynamics. More details can be found from https://www.kaggle.com/datasets/rahulgupta21/datahub-covid19/data.

Free text data based tasks include:

- **T13 Autism spectrum disorder prediction.** This task is built on the TASD dataset, where each sample is a short textual description of a toddler's daily behavior annotated with structured behavioral features such as attention response, word repetition, emotional empathy, noise sensitivity, and more, all derived from early ASD assessment cues. The target is a binary ASD label indicating whether the described behavior corresponds to an ASD or non-ASD toddler. The goal is to learn linguistic and semantic patterns in these behavior-focused narratives so that ML models can detect early ASD-related signals from natural text, supporting more scalable and interpretable early-screening tools. More details can be found from https://www.kaggle.com/datasets/sharafatahmed/tasd-dataset-text-based-early-asd-detection/data.

- **T14 COVID-19 sentiment classification.** This dataset contains thousands of COVID-19–related tweets, each record providing the tweet text along with metadata such as location and timestamp, and the target is a sentiment label manually assigned to reflect the tweet's emotional tone (e.g., positive, negative, neutral, extremely positive, extremely negative) . The task is to train a text-classification model that learns linguistic and contextual cues in the tweets to accurately predict sentiment, enabling analysis of public mood, misinformation patterns, and real-time reactions during the pandemic. More details can be found from https://www.kaggle.com/datasets/datatattle/covid-19-nlp-text-classification/data.

Audio data based tasks include:

- **T15 Respiratory disease prediction.** The goal is to build a multiclass classifier that predicts a disease label for each individual—distinguishing healthy, diseased, and chronic respiratory conditions using the provided audio and metadata. Performance is evaluated using macro F1-score, reflecting the dataset's class imbalance and the need for balanced sensitivity across all disease categories. More details can be found from https://www.kaggle.com/competitions/airs-ai-in-respiratory-sounds/overview.

- **T16 Dysarthria detection.** The task focuses on detecting whether a speech recording comes from a speaker with dysarthria. The data consists of short audio clips paired with minimal speaker information, capturing natural variations in articulation, rhythm, and vocal quality. The model's goal is to learn patterns in these acoustic signals that reliably separate dysarthric from typical speech, without relying on any dataset-specific file structures or formatting details. More details can be found from https://www.kaggle.com/datasets/iamhungundji/dysarthria-detection/data.

Graph data based tasks include:

- **T17 Protein link property prediction.** The ogbl-ddi dataset from the Open Graph Benchmark is designed for link prediction in a large-scale biological interaction graph. It captures a network where nodes represent drugs, and edges represent known drug–drug interactions curated from biomedical knowledge bases. The machine learning task focuses on predicting whether a previously unseen pair of drugs should form an interaction edge, based on the structural patterns of the graph. This makes it a clean, well-defined benchmark for evaluating graph neural networks on real-world pharmacological interaction discovery, with direct relevance to drug safety, polypharmacy risk modeling, and computational drug repurposing. More details can be found from https://ogb.stanford.edu/docs/linkprop/.

A subset of representative task specifications and basic data descriptions for the LLM or agent systems is presented below.

Task 1

```
Task Name: Smoker Status Prediction

[Task Overview]
- Task Type: tabular binary classification
- Modality: structured tabular data (health check / bio-signal related features)
- Downstream Goal: predict whether a person is a smoker based on clinical measurements and basic health indicators.
- Target Column: smoking (0 = non-smoker, 1 = smoker)
- Evaluation Metric: Accuracy
- Objective: train a high-performing model that generalizes well to unseen individuals for this classification task.

[Dataset Paths]
- Dataset Root: /root/Dataset/classifying_smoker_status/
- Train: /root/Dataset/classifying_smoker_status/data/train/train.csv
- Validation: /root/Dataset/classifying_smoker_status/data/val/val.csv
- Test: /root/Dataset/classifying_smoker_status/data/test/test.csv

[Dataset Details]
- Format: CSV
- Columns (brief):
  - ID: sample identifier
  - gender: gender
  - age: age
  - height(cm), weight(kg), waist(cm): basic body measurements
  - eyesight(left/right): eyesight measurements
  - hearing(left/right): hearing measurements
  - systolic, relaxation: blood pressure related measurements
  - fasting blood sugar: fasting blood glucose
  - Cholesterol, triglyceride, HDL, LDL: lipid profile indicators
  - hemoglobin: hemoglobin level
  - Urine protein: urine protein indicator
  - serum creatinine: serum creatinine
  - AST, ALT, Gtp: liver-related indicators
  - oral: oral examination status
  - dental caries: dental caries indicator
  - tartar: tartar status
  - smoking: target label
```

Task 8

```
Task Name: Retinal Blood Vessel Segmentation

[Task Overview]
- Task Type: medical image segmentation (binary segmentation)
- Modality: retinal fundus images with pixel-level vessel masks
- Downstream Goal: segment retinal blood vessels from a fundus image.
- Target: binary mask image (vessel pixels vs background pixels)
- Evaluation Metric: Dice Coefficient
- Objective: train a high-performing segmentation model that produces accurate vessel masks on unseen retinal images.

[Dataset Paths]
- Dataset Root: /root/Dataset/Retina_Blood_Vessel_for_segmentation/
- Train Images: /root/Dataset/Retina_Blood_Vessel_for_segmentation/data/train/image/
```

```
- Train Masks: /root/Dataset/Retina_Blood_Vessel_for_segmentation/data/train/mask/
- Validation Images: /root/Dataset/Retina_Blood_Vessel_for_segmentation/data/val/image/
- Validation Masks: /root/Dataset/Retina_Blood_Vessel_for_segmentation/data/val/mask/
- Test Images: /root/Dataset/Retina_Blood_Vessel_for_segmentation/data/test/image/
- Test Masks: /root/Dataset/Retina_Blood_Vessel_for_segmentation/data/test/mask/

[Dataset Details]
- Format: PNG images
- Paired files (brief):
  - image/: retinal fundus images
  - mask/: corresponding binary vessel masks with the same filename
```

Task 15

```
Task Name: Respiratory Disease Prediction

[Task Overview]
- Task Type: multi-modal classification (tabular + audio)
- Modality: structured tabular metadata + respiratory audio recordings (cough and vowel)
- Downstream Goal: predict the respiratory disease class for each candidate based on metadata and audio signals.
- Target Column: disease (categorical label)
- Evaluation Metric: Macro-F1
- Objective: train a high-performing model that generalizes well to unseen candidates for robust respiratory disease
detection.

[Dataset Paths]
- Dataset Root: /root/Dataset/medical_sound_classification/
- Train Table: /root/Dataset/medical_sound_classification/data/train/train.csv
- Train Sounds: /root/Dataset/medical_sound_classification/data/train/sounds/
- Validation Table: /root/Dataset/medical_sound_classification/data/val/val.csv
- Validation Sounds: /root/Dataset/medical_sound_classification/data/val/sounds/
- Test Table: /root/Dataset/medical_sound_classification/data/test/test.csv
- Test Sounds: /root/Dataset/medical_sound_classification/data/test/sounds/

[Dataset Details]
- Format: CSV (tables) + WAV (audio) + JSON (precomputed embeddings)
- Table columns (brief):
  - candidateID: candidate identifier (maps to the corresponding audio folder)
  - age, gender: basic demographics
  - tbContactHistory, wheezingHistory, phlegmCough, familyAsthmaHistory, feverHistory, coldPresent: symptom / history
indicators
  - packYears: smoking exposure indicator
  - disease: target label
- Audio/embedding files (brief):
  - Each candidate folder contains cough.wav, vowel.wav, and optional embedding files (emb_cough.json, emb_vowel.json).
```

A.2 Baseline

Prompt for LLM

```
You are a senior machine learning engineer. Your objective is to build a model with optimal predictive performance and to
select an appropriate uncertainty estimation method to integrate with the model, such that uncertainty-related metrics (e.
g., ECE, Brier Score, PCIP) can be computed and reported. Based on the user's task description, write a complete Python
script.

Requirements:
1. The script must load the data directly from the provided file paths.
2. The script must compute and print all evaluation metrics specified by the task.
3. The code must be enclosed within python and blocks.
4. Do not use the input() function; the script must run fully automatically.

In addition to completing the modeling task, select an appropriate uncertainty estimation method and output the
corresponding uncertainty analysis results.
```

Prompt for Claude Code

```
Please read the task file task description carefully and strictly follow all instructions without asking for clarification.


Your task is to train a model that achieves optimal predictive performance and to select an appropriate uncertainty
estimation method to integrate with the model, such that uncertainty-related metrics (e.g., ECE, Brier Score,) can be
computed and reported.

Before starting each task, create a corresponding task directory under 'OUTPUTROOT', named using the task name and a
timestamp.

All outputs and execution scripts for each task must be placed in the corresponding directory. After completion, provide a
 brief summary of the executed steps and results, and must report both model performance metrics and uncertainty analysis
results. If the task fails, explicitly report the failure. The final output should be written to a file named result.txt.

Do not ask for confirmation. Directly execute file writing and command execution. All code should be run directly by you.
```

B Additional Details for Results

This section presents the detailed experimental results. Figure 3 is generated from three-run averages, and Tables 13 and 17 report the corresponding task-level normalized prediction and uncertainty scores. Table 10 further summarizes task-level model-execution and uncertainty-analysis success rates.

Task Implementation Across repeated runs, *AutoHealth* maintains the strongest task-completion behavior, with an average model-execution success rate of 98.0% and an average uncertainty-analysis success rate of 98.0%. *Claude Code* also completes many tasks but is less stable under repeated execution, whereas the naïve LLM, *DS-Agent*, and *AutoML-Agent* fail frequently during either model construction or uncertainty quantification. These failures are most pronounced on heterogeneous non-tabular tasks such as IMU-based gesture prediction, EEG prediction, respiratory audio classification, dysarthria detection, and graph link prediction.

Token and Cost Analysis. Table 9 presents the token usage, monetary cost, and wall-clock time across all tasks, and Figure 5 displays the execution-time proportion of each stage. Across the 17 tasks, the total token consumption remains within a moderate range, while the monetary cost is consistently low, typically below 6 RMB per task. Importantly, *AutoHealth* demonstrates fast iterative cycles, with most tasks completing within 2–6 hours (5 runs), even for tasks involving gigabyte-scale data. This efficiency is largely attributed to effective and structured multi-agent coordination, which substantially reduces redundant token consumption across iterations. By integrating task understanding, model construction, execution, and feedback into a unified end-to-end agentic workflow, *AutoHealth* minimizes human intervention and iteration overhead. Together, these results indicate that *AutoHealth* enables rapid and affordable end-to-end experimentation, thereby lowering the barrier to large-scale empirical studies and accelerating scientific discovery in real-world health data modeling.

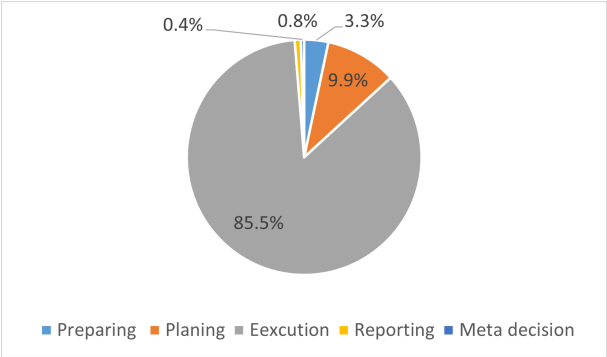


Fig. 5: Execution time proportion of each stage.

Table 9: Token and time cost across all tasks. Input, output and cache token counts are reported in millions (M).

Task	Data size	Input	Output	Cache	Cost (RMB)	Time (min)
T1	11.03MB	1.58	0.20	0.93	2.1	119.8
T2	31.41MB	2.24	0.27	1.39	2.8	156.9
T3	215.96MB	2.60	0.35	1.47	3.6	470.9
T4	35.46MB	2.01	0.23	1.27	2.4	123.0
T5	9.87MB	2.65	0.38	1.49	3.8	246.1
T6	143.5MB	3.49	0.55	1.76	5.5	478.7
T7	83.27MB	3.28	0.40	1.78	4.6	360.2
T8	34.76MB	4.05	0.65	1.98	6.5	344.1
T9	7.75MB	1.81	0.22	1.14	2.2	123.2
T10	1.12GB	3.40	0.38	1.97	4.4	214.4
T11	6.6GB	3.73	0.47	1.83	5.6	634.1
T12	3.18MB	1.83	0.20	1.11	2.3	109.8
T13	57.15KB	1.72	0.24	1.04	2.3	231.2
T14	11.5MB	3.21	0.39	1.88	4.2	332.3
T15	1.9GB	3.03	0.38	1.69	4.2	187.5
T16	200MB	2.80	0.38	1.66	3.8	220.6
T17	102MB	3.38	0.47	1.77	5.0	419.5

Downstream Performance The following tables provide the detailed evidence chain for the repeated-run results reported in the main text. Table 10 reports model-execution and uncertainty-analysis success

rates. Tables 11–18 report raw and normalized prediction and uncertainty results, with means and standard deviations shown separately. Table 19 reports the task-level SR, NPS, and CS used for Figure 3. Table 20 reports the full-task single-agent attribution details. Table 21 summarizes the backbone-robustness calculation, and Tables 22–23 report the two full-task uncertainty-ablation tables.

Table 10: Model-execution and uncertainty-analysis success rates across three repeated runs. Values denote the fraction of successful runs for each task.

Model-execution success rate																		
Method	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	AVG
LLM	0.000	0.000	0.000	0.667	0.000	0.000	0.333	0.000	0.333	0.000	0.000	0.333	0.667	0.333	0.000	0.000	0.000	0.157
DS-Agent	0.667	0.667	0.000	0.667	0.667	0.000	0.000	0.000	0.333	0.000	0.000	0.667	1.000	0.667	0.000	0.000	0.000	0.314
AutoML-Agent	0.667	0.667	0.667	1.000	1.000	0.667	1.000	0.000	0.000	0.000	0.000	0.667	1.000	1.000	0.000	0.000	0.000	0.490
Claude Code	1.000	1.000	0.667	1.000	1.000	1.000	1.000	1.000	1.000	0.667	1.000	1.000	1.000	1.000	1.000	1.000	0.667	0.941
AutoHealth	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.667**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.980**

Uncertainty-analysis success rate																		
Method	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	AVG
LLM	0.000	0.000	0.000	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.000	0.000	0.000	0.000	0.039
DS-Agent	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AutoML-Agent	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Claude Code	1.000	1.000	0.667	1.000	0.333	1.000	1.000	1.000	1.000	0.333	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.902
AutoHealth	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.667**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.980**

Table 11: Raw prediction performance means across three repeated runs. Fail denotes unsuccessful runs and N/A denotes unsupported tasks.

Method	T1 Acc	T2 Acc	T3 LogLoss	T4 RMSLE	T5 C-index	T6 F1	T7 Acc	T8 Dice	T9 F1	T10 Av-F1	T11 KL	T12 MAE	T13 Acc	T14 Acc	T15 Macro-F1	T16 Acc	T17 Hits@20
LLM	Fail	Fail	Fail	0.084	Fail	Fail	0.784	Fail	0.848	Fail	Fail	0.827	0.820	0.571	Fail	Fail	Fail
DS-Agent	0.701	0.860	Fail	0.065	0.582	Fail	Fail	Fail	0.862	Fail	Fail	0.696	0.823	0.561	Fail	Fail	Fail
AutoML-Agent	0.726	0.925	0.320	0.067	0.638	0.334	0.825	N/A	Fail	Fail	Fail	0.820	0.861	0.807	N/A	N/A	N/A
Claude Code	0.694	0.800	0.183	0.073	0.626	0.316	0.514	0.668	0.933	0.398	1.050	0.772	0.886	0.675	0.691	0.618	0.201
AutoHealth	**0.757**	**0.903**	**0.081**	**0.062**	**0.642**	**0.484**	**0.827**	**0.770**	**0.862**	**0.446**	**0.791**	**0.160**	**0.937**	**0.796**	**0.737**	**0.768**	**0.858**

Table 12: Raw prediction performance standard deviations across three repeated runs. Fail denotes unsuccessful runs, N/A denotes unsupported tasks, and a dash indicates that the standard deviation is unavailable.

Method	T1 Acc	T2 Acc	T3 LogLoss	T4 RMSLE	T5 C-index	T6 F1	T7 Acc	T8 Dice	T9 F1	T10 Av-F1	T11 KL	T12 MAE	T13 Acc	T14 Acc	T15 Macro-F1	T16 Acc	T17 Hits@20
LLM	Fail	Fail	Fail	0.001	Fail	Fail	–	Fail	–	Fail	Fail	–	0.052	–	Fail	Fail	Fail
DS-Agent	0.025	0.032	Fail	0.001	0.028	Fail	Fail	Fail	–	Fail	Fail	0.023	0.023	0.042	Fail	Fail	Fail
AutoML-Agent	0.112	0.002	0.278	0.005	0.034	0.021	0.030	N/A	Fail	Fail	Fail	0.021	0.046	0.052	N/A	N/A	N/A
Claude Code	0.108	0.041	0.021	0.014	0.010	0.026	0.024	0.119	0.018	–	0.202	0.097	0.050	0.122	0.100	0.032	0.020
AutoHealth	**0.036**	**0.021**	**0.018**	**0.000**	**0.003**	**0.206**	**0.237**	**0.055**	**0.014**	**0.190**	**0.049**	**0.091**	**0.048**	**0.025**	**0.140**	**0.075**	**0.112**

Table 13: Normalized prediction score means across three repeated runs. Values range from 0 to 1; higher is better.

Method	T1 Acc	T2 Acc	T3 LogLoss	T4 RMSLE	T5 C-index	T6 F1	T7 Acc	T8 Dice	T9 F1	T10 Av-F1	T11 KL	T12 MAE	T13 Acc	T14 Acc	T15 Macro-F1	T16 Acc	T17 Hits@20	AVG
LLM	0.000	0.000	0.000	0.362	0.000	0.000	0.261	0.000	0.283	0.000	0.000	0.036	0.547	0.190	0.000	0.000	0.000	0.099
DS-Agent	0.467	0.573	0.000	0.405	0.388	0.000	0.000	0.000	0.287	0.000	0.000	0.084	0.823	0.374	0.000	0.000	0.000	0.200
AutoML-Agent	0.484	0.617	0.204	0.599	0.638	0.223	0.825	0.000	0.000	0.000	0.000	0.072	0.861	0.806	0.000	0.000	0.000	0.313
Claude Code	0.694	0.800	0.236	0.581	0.626	0.316	0.514	0.668	0.933	0.133	0.089	0.116	0.886	0.675	0.691	0.618	0.134	0.512
AutoHealth	**0.757**	**0.903**	**0.557**	**0.618**	**0.642**	**0.484**	**0.827**	**0.770**	**0.862**	**0.446**	**0.075**	**0.416**	**0.937**	**0.796**	**0.737**	**0.768**	**0.858**	**0.674**

Table 14: Normalized prediction score standard deviations across three repeated runs.

Method	T1 Acc	T2 Acc	T3 LogLoss	T4 RMSLE	T5 C-index	T6 F1	T7 Acc	T8 Dice	T9 F1	T10 Av-F1	T11 KL	T12 MAE	T13 Acc	T14 Acc	T15 Macro-F1	T16 Acc	T17 Hits@20	AVG
LLM	0.000	0.000	0.000	0.314	0.000	0.000	0.453	0.000	0.490	0.000	0.000	0.062	0.475	0.330	0.000	0.000	0.000	0.125
DS-Agent	0.405	0.497	0.000	0.351	0.337	0.000	0.000	0.000	0.498	0.000	0.000	0.073	0.023	0.325	0.000	0.000	0.000	0.148
AutoML-Agent	0.427	0.534	0.227	0.016	0.034	0.194	0.030	0.000	0.000	0.000	0.000	0.063	0.046	0.052	0.000	0.000	0.000	0.095
Claude Code	0.108	0.041	0.205	0.046	0.010	0.026	0.024	0.119	0.018	0.229	0.017	0.012	0.050	0.122	0.100	0.032	0.117	0.075
AutoHealth	**0.036**	**0.021**	**0.054**	**0.000**	**0.003**	**0.206**	**0.237**	**0.055**	**0.014**	**0.190**	**0.065**	**0.134**	**0.048**	**0.025**	**0.140**	**0.075**	**0.112**	**0.083**

Table 15: Raw uncertainty-analysis means across three repeated runs. Lower values are better for BS, ECE, and ΔPCIP; Fail denotes unsuccessful uncertainty analysis and N/A denotes unsupported uncertainty output.

Method	T1 BS	T2 BS	T3 BS	T4 ΔPCIP	T5 ΔPCIP	T6 ECE	T7 ECE	T8 BS	T9 ECE	T10 ECE	T11 ECE	T12 ΔPCIP	T13 ECE	T14 ECE	T15 ECE	T16 ECE	T17 BS	AVG
LLM	Fail	Fail	Fail	0.018	Fail	Fail	Fail	Fail	Fail	Fail	Fail	0.505	Fail	Fail	Fail	Fail	Fail	0.262
DS-Agent	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	–
AutoML-Agent	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	–
Claude Code	0.152	0.128	0.030	0.227	0.889	0.265	0.244	0.100	0.023	0.116	0.174	0.352	0.059	0.211	0.196	0.241	0.138	0.209
AutoHealth	**0.036**	**0.075**	**0.013**	**0.003**	**0.008**	**0.049**	**0.009**	**0.044**	**0.097**	**0.158**	**0.190**	**0.225**	**0.055**	**0.070**	**0.080**	**0.133**	**0.252**	**0.088**

Table 16: Raw uncertainty-analysis standard deviations across three repeated runs. Fail denotes unsuccessful uncertainty analysis, and a dash indicates that the standard deviation is unavailable.

Method	T1 BS	T2 BS	T3 BS	T4 ΔPCIP	T5 ΔPCIP	T6 ECE	T7 ECE	T8 BS	T9 ECE	T10 ECE	T11 ECE	T12 ΔPCIP	T13 ECE	T14 ECE	T15 ECE	T16 ECE	T17 BS	AVG
LLM	Fail	Fail	Fail	–	Fail	Fail	Fail	Fail	Fail	Fail	Fail	–	Fail	Fail	Fail	Fail	Fail	–
DS-Agent	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	–
AutoML-Agent	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	–
Claude Code	0.126	0.022	0.014	0.099	–	0.045	0.059	0.016	0.232	–	0.036	0.044	0.100	0.194	0.017	0.005	0.041	–
AutoHealth	**0.095**	**0.005**	**0.025**	**0.009**	**0.002**	**0.037**	**0.110**	**0.031**	**0.035**	**0.070**	**0.039**	**0.016**	**0.046**	**0.017**	**0.069**	**0.026**	**0.065**	–

Table 17: Normalized uncertainty score means across three repeated runs. Values range from 0 to 1; higher is better.

Method	T1 BS	T2 BS	T3 BS	T4 ΔPCIP	T5 ΔPCIP	T6 ECE	T7 ECE	T8 BS	T9 ECE	T10 ECE	T11 ECE	T12 ΔPCIP	T13 ECE	T14 ECE	T15 ECE	T16 ECE	T17 BS	AVG
LLM	0.000	0.000	0.000	0.282	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.055	0.000	0.000	0.000	0.000	0.000	0.020
DS-Agent	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AutoML-Agent	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Claude Code	0.341	0.487	0.479	0.530	0.034	0.251	0.324	0.523	0.572	0.154	0.424	0.208	0.483	0.227	0.341	0.299	0.415	0.358
AutoHealth	**0.470**	**0.554**	**0.813**	**0.894**	**0.918**	**0.586**	**0.714**	**0.610**	**0.451**	**0.364**	**0.212**	**0.293**	**0.501**	**0.584**	**0.444**	**0.491**	**0.321**	**0.542**

Table 18: Normalized uncertainty score standard deviations across three repeated runs.

Method	T1 BS	T2 BS	T3 BS	T4 ΔPCIP	T5 ΔPCIP	T6 ECE	T7 ECE	T8 BS	T9 ECE	T10 ECE	T11 ECE	T12 ΔPCIP	T13 ECE	T14 ECE	T15 ECE	T16 ECE	T17 BS	AVG
LLM	0.000	0.000	0.000	0.489	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.095	0.000	0.000	0.000	0.000	0.000	0.034
DS-Agent	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AutoML-Agent	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Claude Code	0.109	0.054	0.418	0.196	0.058	0.027	0.068	0.045	0.337	0.267	0.065	0.018	0.175	0.090	0.020	0.005	0.068	0.119
AutoHealth	**0.233**	**0.016**	**0.143**	**0.073**	**0.015**	**0.111**	**0.331**	**0.103**	**0.065**	**0.080**	**0.185**	**0.014**	**0.126**	**0.055**	**0.119**	**0.060**	**0.073**	**0.106**

Table 19: Task-level SR, NPS, and CS used for Figure 3. AVG denotes the average across all 17 tasks.

SR

Method	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	AVG
LLM	0.000	0.000	0.000	0.500	0.000	0.000	0.167	0.000	0.167	0.000	0.000	0.333	0.333	0.167	0.000	0.000	0.000	0.098
DS-Agent	0.333	0.333	0.000	0.333	0.333	0.000	0.000	0.000	0.167	0.000	0.000	0.333	0.500	0.333	0.000	0.000	0.000	0.157
AutoML-Agent	0.333	0.333	0.333	0.500	0.500	0.333	0.500	0.000	0.000	0.000	0.000	0.333	0.500	0.500	0.000	0.000	0.000	0.245
Claude Code	1.000	1.000	0.667	1.000	0.667	1.000	1.000	1.000	1.000	0.500	1.000	1.000	1.000	1.000	1.000	1.000	0.833	0.922
AutoHealth	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.667**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.980**

NPS

Method	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	AVG
LLM	0.000	0.000	0.000	0.322	0.000	0.000	0.131	0.000	0.141	0.000	0.000	0.045	0.274	0.095	0.000	0.000	0.000	0.059
DS-Agent	0.234	0.286	0.000	0.203	0.194	0.000	0.000	0.000	0.143	0.000	0.000	0.042	0.411	0.187	0.000	0.000	0.000	0.100
AutoML-Agent	0.242	0.308	0.102	0.299	0.319	0.112	0.412	0.000	0.000	0.000	0.000	0.036	0.430	0.403	0.000	0.000	0.000	0.157
Claude Code	0.517	0.643	0.357	0.555	0.330	0.283	0.419	0.596	0.752	0.144	0.257	0.162	0.684	0.451	0.516	0.459	0.274	0.435
AutoHealth	**0.613**	**0.729**	**0.685**	**0.756**	**0.780**	**0.535**	**0.770**	**0.690**	**0.656**	**0.405**	**0.143**	**0.354**	**0.719**	**0.690**	**0.591**	**0.629**	**0.590**	**0.608**

CS

Method	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	AVG
LLM	0.000	0.000	0.000	0.411	0.000	0.000	0.149	0.000	0.154	0.000	0.000	0.189	0.303	0.131	0.000	0.000	0.000	0.079
DS-Agent	0.283	0.310	0.000	0.268	0.264	0.000	0.000	0.000	0.155	0.000	0.000	0.188	0.456	0.260	0.000	0.000	0.000	0.128
AutoML-Agent	0.288	0.321	0.218	0.400	0.409	0.222	0.456	0.000	0.000	0.000	0.000	0.185	0.465	0.452	0.000	0.000	0.000	0.201
Claude Code	0.759	0.822	0.512	0.778	0.498	0.642	0.710	0.798	0.876	0.322	0.628	0.581	0.842	0.726	0.758	0.729	0.554	0.678
AutoHealth	**0.807**	**0.864**	**0.843**	**0.878**	**0.890**	**0.767**	**0.885**	**0.845**	**0.828**	**0.703**	**0.405**	**0.677**	**0.860**	**0.845**	**0.795**	**0.815**	**0.795**	**0.794**

Table 20: Full-task single-agent attribution details using the same task-level metrics as Table 19.

Method	Metric	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	AVG
Single-agent	SR	1.000	1.000	0.834	1.000	0.500	0.834	0.834	1.000	1.000	0.000	0.000	0.834	1.000	0.834	0.667	0.834	0.834	0.765
	NPS	0.583	0.606	0.450	0.460	0.270	0.239	0.471	0.483	0.657	0.000	0.000	0.113	0.640	0.479	0.327	0.457	0.195	0.378
	CS	0.791	0.803	0.641	0.730	0.385	0.536	0.652	0.742	0.829	0.000	0.000	0.473	0.820	0.656	0.497	0.645	0.514	0.571
AutoHealth	SR	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.667**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.980**
	NPS	**0.613**	**0.729**	**0.685**	**0.756**	**0.780**	**0.535**	**0.770**	**0.690**	**0.656**	**0.405**	**0.143**	**0.354**	**0.719**	**0.690**	**0.591**	**0.629**	**0.590**	**0.608**
	CS	**0.807**	**0.864**	**0.843**	**0.878**	**0.890**	**0.767**	**0.885**	**0.845**	**0.828**	**0.703**	**0.405**	**0.677**	**0.860**	**0.845**	**0.795**	**0.815**	**0.795**	**0.794**

Table 21: Backbone robustness calculation summary on five representative tasks. Values report mean $\pm$ std after task-specific normalization.

Backbone	AutoHealth Pred.	Claude Code Pred.	AutoHealth Unc.	Claude Code Unc.
DeepSeek-V3.2	**0.643±0.069**	0.592±0.094	**0.636±0.067**	0.382±0.018
DeepSeek-V3	**0.578±0.086**	0.523±0.246	**0.557±0.100**	0.447±0.077
Gemini-3-flash	**0.675±0.027**	0.654±0.062	**0.627±0.073**	0.409±0.073
Gemini-2.0-flash	**0.480±0.207**	0.187±0.094	**0.225±0.059**	0.000±0.000
Qwen3.5-27B	**0.640±0.027**	0.613±0.019	**0.600±0.016**	0.548±0.024

Table 22: Full-task uncertainty ablation with normalized scores.

Method	T1 BS	T2 BS	T3 BS	T4 ΔPCIP	T5 ΔPCIP	T6 ECE	T7 ECE	T8 BS	T9 ECE	T10 ECE	T11 ECE	T12 ΔPCIP	T13 ECE	T14 ECE	T15 ECE	T16 ECE	T17 BS	AVG
Claude Code	0.397	0.440	0.769	0.306	0.101	0.274	0.291	0.500	0.813	0.463	0.365	0.221	0.629	0.322	0.338	0.293	0.420	0.408
AutoHealth	**0.738**	**0.572**	**0.887**	**0.971**	**0.929**	**0.671**	**0.917**	**0.694**	**0.508**	**0.388**	**0.345**	**0.308**	**0.645**	**0.588**	**0.556**	**0.429**	**0.284**	**0.614**
AutoHealth-T	0.587	0.513	0.887	0.806	0.873	0.530	0.864	0.654	0.467	0.321	0.279	0.241	0.547	0.518	0.501	0.351	0.267	0.542
AutoHealth-R	0.410	0.561	0.806	0.746	0.725	0.417	0.725	0.623	0.434	0.341	0.278	0.253	0.329	0.426	0.361	0.280	0.271	0.470

Table 23: Full-task uncertainty ablation with back-calculated raw uncertainty values. Raw values are derived from normalized scores using $s = 0.1/S - 0.1$; lower is better.

Method	T1 BS	T2 BS	T3 BS	T4 ΔPCIP	T5 ΔPCIP	T6 ECE	T7 ECE	T8 BS	T9 ECE	T10 ECE	T11 ECE	T12 ΔPCIP	T13 ECE	T14 ECE	T15 ECE	T16 ECE	T17 BS	AVG
Claude Code	0.152	0.127	0.030	0.227	0.890	0.265	0.244	0.100	0.023	0.116	0.174	0.352	0.059	0.211	0.196	0.241	0.138	0.208
AutoHealth	**0.035**	**0.075**	**0.013**	**0.003**	**0.008**	**0.049**	**0.009**	**0.044**	**0.097**	**0.158**	**0.190**	**0.225**	**0.055**	**0.070**	**0.080**	**0.133**	**0.252**	**0.088**
AutoHealth-T	0.070	0.095	0.013	0.024	0.015	0.089	0.016	0.053	0.114	0.211	0.258	0.315	0.083	0.093	0.100	0.185	0.275	0.118
AutoHealth-R	0.144	0.078	0.024	0.034	0.038	0.140	0.038	0.060	0.130	0.193	0.260	0.295	0.204	0.135	0.177	0.257	0.269	0.146

C Additional Details for Report Quality

C.1 Additional Details for LLM-as-a-judge

Prompt for LLM

```
You are an experienced clinician and clinical AI reviewer with expertise in medical machine learning, trustworthy AI,
uncertainty quantification, and deployment of AI systems in healthcare. You are given an automatically generated machine
learning report for a healthcare prediction task. Your job is to evaluate the report from the perspective of:

Clinical usefulness

Scientific rigor

Reliability and calibration

Deployment readiness

Safety and fairness awareness

Readability for clinicians

You must score the report quantitatively using the evaluation framework below.

IMPORTANT:

Evaluate the REPORT QUALITY, not only the model performance.

Focus on whether a clinician could understand, trust, and potentially use the system.

Penalize overconfident claims, lack of calibration discussion, missing limitations, or poor deployment reasoning.

Reward uncertainty awareness, clinically meaningful interpretation, safety discussion, and actionable recommendations.

Be strict and critical like a reviewer for Nature Medicine / NEJM AI / Lancet Digital Health.

For each metric:

Give a score from 1 to 5

Provide a short justification

Use the following interpretation:

1 = Very poor
2 = Weak
3 = Acceptable
4 = Strong
5 = Excellent

SECTION A  Clinical Readability
A1. Clinical Readability
Can clinicians understand the report without advanced ML expertise?

Consider:

clarity of language

excessive jargon

sentence complexity

logical organization

readability of conclusions

A2. Explanation Completeness
Does the report sufficiently explain:

task definition

dataset

preprocessing

model choice

metrics

uncertainty

limitations

deployment implications

A3. Actionability
Does the report provide clinically actionable insights?

Examples:

screening vs diagnosis

threshold implications

uncertainty triage
```

```
referral recommendations

workflow suggestions

A4. Cognitive Burden
Is the report concise and cognitively manageable for clinicians?

Penalize:

unnecessarily long explanations

repetitive text

overly technical derivations

unclear structure

SECTION B  Clinical Reliability
B1. Calibration Awareness
Does the report appropriately discuss probability calibration?

Look for:

ECE

Brier score

NLL

calibration curves

pre/post calibration discussion

interpretation of calibrated probabilities

B2. Uncertainty Quality
Does the report meaningfully quantify and interpret uncertainty?

Look for:

uncertainty metrics

uncertainty-error relationship

selective prediction

abstention logic

uncertainty triage

subgroup uncertainty analysis

B3. Clinical Safety Awareness
Does the report acknowledge risks and limitations?

Examples:

not suitable for autonomous diagnosis

need for human oversight

deployment cautions

uncertainty limitations

subgroup risks

B4. Fairness and Bias Awareness
Does the report discuss:

subgroup behavior

demographic bias

shortcut learning

fairness limitations

subgroup calibration

SECTION C  Deployment Readiness
C1. Deployment Readiness
Could this report realistically support clinical deployment discussions?

Consider:

workflow integration

operational interpretation

threshold selection

triage strategy

risk stratification

clinical use-case clarity
```

```
C2. Human-in-the-Loop Design
Does the report appropriately integrate clinician oversight?

Examples:

manual review for uncertain cases

selective automation

escalation logic

clinician confirmation

C3. External Validity Awareness
Does the report discuss:

external validation

domain shift

demographic shift

hospital/site variation

generalizability limitations

SECTION D  Scientific Rigor
D1. Methodological Transparency
Does the report clearly explain:

preprocessing

feature engineering

model selection

validation strategy

leakage prevention

D2. Evidence-grounded Reasoning
Are claims appropriately supported by evidence and metrics?

Penalize:

exaggerated deployment claims

unsupported statements

hallucinated conclusions

D3. Limitation Quality
Does the report critically discuss limitations?

Examples:

calibration limitations

threshold dependence

subgroup weakness

uncertainty limitations

lack of external validation

SECTION E  Overall Assessment
E1. Overall Clinical Usefulness
Overall, how useful would this report be to a clinician?

E2. Overall Trustworthiness
How trustworthy and responsible is the report?

E3. Publication-level Quality
Could this style of report appear in a serious clinical AI paper?

Return the evaluation in JSON format.

Example:

{
"Clinical Readability": {
"Clinical Readability": {
"score": 4,
"justification": "The report is clearly written and avoids excessive ML jargon."
},
...
},

"Clinical Reliability": {
...
},

"Deployment Readiness": {
...
},

"Scientific Rigor": {
```

```
...
},

"Overall Assessment": {
"Overall Clinical Usefulness": {
"score": 4,
"justification": "Useful for screening-oriented decision support."
},
...
},

"summary": {
"major_strengths": [
"...",
"..."
],
"major_weaknesses": [
"...",
"..."
],
"deployment_readiness": "Low / Moderate / High",
"overall_recommendation": "..."
}
}

[ATTACH REPORT HERE]
```

C.2 Generated Report for Task 15

Respiratory Disease Prediction from Multi-modal Data: A Report for Data, Model, Performance and Uncertainty Analysis

by AutoHealth

Abstract

This report details the experiments conducted by AutoHealth, a multi-agent system designed for antonymous health data modeling. This task is to predict respiratory disease using a combination of structured tabular metadata (demographics, symptoms, history) and respiratory audio recordings (cough and vowel sounds). A dual-branch late-fusion neural network was developed, integrating features from both modalities. To enhance robustness and provide uncertainty estimates, a deep ensemble of three models was trained with controlled architectural variations. The final model achieved a Macro-F1 score of 0.8384 on the test set, demonstrating strong generalization. Comprehensive uncertainty analysis revealed that incorrect predictions are associated with higher model uncertainty, providing a valuable tool for identifying low-confidence cases in clinical deployment. The report includes a detailed data analysis, model training methodology, performance evaluation, and actionable insights for clinical application.

1 Data Analysis

1.1 Dataset Overview

The dataset is a multi-modal collection designed for respiratory disease classification. It comprises 546 unique candidates, split into training (437), validation (54), and test (55) sets. Each candidate is represented by two data modalities:

- **Tabular Metadata**: An 11-column CSV file containing demographic information (age, gender), symptom/history indicators (e.g., wheezingHistory, phlegmCough, familyAsthmaHistory), smoking exposure (packYears), and the target label (disease: Class 0, 1, or 2).
- **Audio Recordings**: Two WAV files per candidate: a cough recording (avg. 16.0s) and a vowel recording (avg. 6.1s), both sampled at 48 kHz. Precomputed audio embeddings (JSON files) are also provided, representing deep features extracted from the audio signals.

The primary evaluation metric is Macro-F1, which is suitable for handling the observed class imbalance.

1.2 Feature Analysis and Clinical Correlations

Analysis revealed strong predictive signals within the data, aligning with clinical intuition.

- **Tabular Features**: Age and packYears (smoking exposure) showed high point-biserial correlation with the disease label (0.567 and 0.372, respectively). Categorical features like wheezingHistory and phlegmCough were also highly significant (Chi-square p $<$ 1e-07). For instance, Class 1 had an 84.7% positive rate for wheezingHistory, compared to only 5.3% for Class 0.
- **Audio Features**: After mean-pooling the variable-length cough embeddings, 206 out of 512 feature dimensions showed significant inter-class differences (ANOVA $p < 0.05$). These audio features exhibited low correlation with strong tabular features ($|r| < 0.1$), suggesting they provide complementary, non-redundant information for classification.

1.3 Data Quality and Challenges

Several data quality issues were identified and addressed in the preprocessing pipeline:

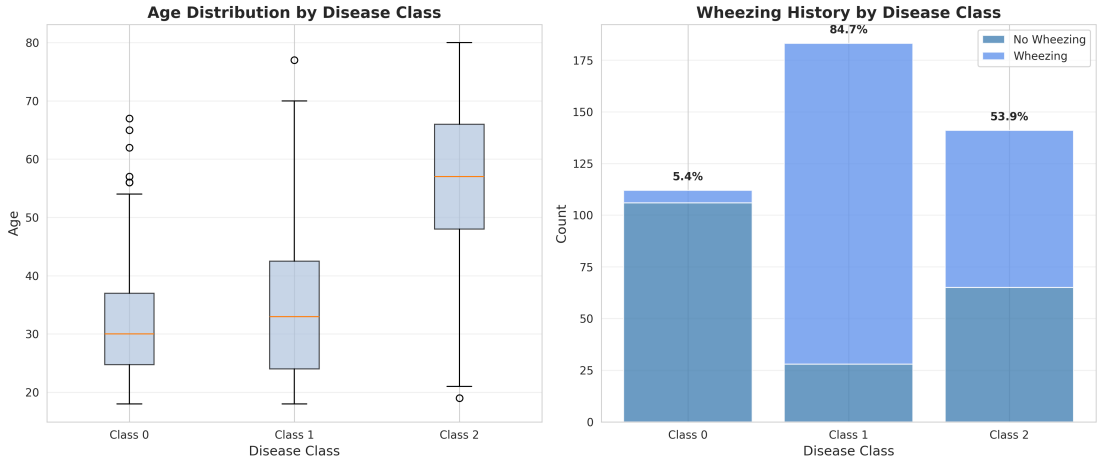


Figure 1: Relationship between key features and the target disease class. **Left:** Age distribution across disease classes, showing Class 0 as youngest and Class 2 as oldest. **Right:** Prevalence of wheezing history within each class, highlighting its strong association with Class 1. These features were prioritized during model design.

- **Missing Values**: The ‘coldPresent‘ column contained significant missing values (115 in train set). These were imputed using the mode from the training data.
- **Class Imbalance**: The target distribution is uneven. In the training set, Class 1 is most frequent (183 samples), followed by Class 2 (141), and Class 0 (113). This imbalance was actively mitigated during training.
- **Audio Embedding Inconsistency**: The precomputed embeddings had variable sequence lengths (e.g., (1,512), (4,512)). A global average pooling operation was applied to standardize them into fixed-length vectors.
- **Distribution Shift**: Kolmogorov-Smirnov tests indicated a mild distribution shift in some audio embedding dimensions between the training and validation/test sets, necessitating careful feature scaling.

2 Model Training

2.1 Data Preprocessing

A comprehensive preprocessing pipeline was implemented to prepare the multi-modal data:

- **Tabular Data**: Categorical variables were label-encoded. Missing values in ‘coldPresent‘ were filled with the training set mode. A simple interaction feature, ‘elderly_with_wheezing‘, was created. Numerical features (‘age‘, ‘packYears‘) were standardized using the training set’s mean and standard deviation.
- **Audio Data**: For each candidate, the ‘emb_cough.json‘ and ‘emb_vowel.json‘ files were loaded. The variable-length sequences were mean-pooled into 512-dimensional vectors each, then concatenated to form a 1024-dimensional audio feature vector, which was subsequently standardized.
- **Class Imbalance Mitigation**: Two strategies were employed: 1) Class weights ([1.25, 0.77, 1.00]) were calculated for the loss function to penalize errors on minority classes more heavily. 2) A ‘WeightedRandomSampler‘ was used during training to ensure each mini-batch had a balanced representation of all three classes.
- **Data Augmentation**: To improve generalization, Gaussian noise was added to both tabular and audio feature vectors during training (std=0.03 and 0.02, respectively).

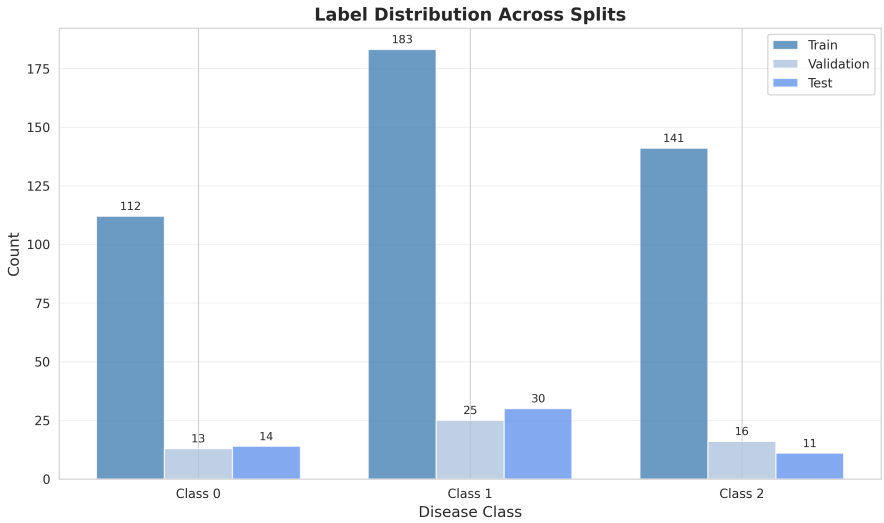


Figure 2: Label distribution across training, validation, and test splits. The class imbalance (Class 1 > Class 2 > Class 0) is consistent across splits, justifying the use of class-weighted loss and balanced sampling strategies during model training to prevent bias.

2.2 Model Architecture

A custom dual-branch late-fusion neural network was designed to effectively combine information from both modalities.

- **Audio Branch**: A Multi-Layer Perceptron (MLP) takes the 1024-dim audio vector, projects it through layers (1024→512→256) with BatchNorm and Dropout, outputting a 256-dim representation.
- **Tabular Branch**: A separate MLP processes the 10-dim tabular vector through layers (10→64→32), outputting a 32-dim representation.
- **Fusion & Classification**: The outputs from both branches are concatenated (288-dim). A final classification head (MLP: 288→128→64→3) produces the logits for the three disease classes.

This architecture allows each branch to learn specialized representations before combining them for the final decision.

2.3 Hyperparameters

Key hyperparameters were selected to balance learning capacity and regularization:

- **Optimizer**: AdamW with initial learning rate $lr = 5 \times 10^{-4}$ and weight decay 5×10^{-4}.
- **Learning Rate Schedule**: ReduceLROnPlateau scheduler (factor=0.5, patience=10 epochs).
- **Regularization**: Dropout rates of 0.4 in key layers, combined with BatchNorm, were used to prevent overfitting.
- **Training**: Batch size of 32 for up to 80 epochs, with early stopping (patience=20) based on validation loss.
- **Ensemble**: Three independent models were trained with different random seeds (42, 123, 456) and slight variations in audio branch dropout rates (0.35, 0.4, 0.45) to introduce diversity.

2.4 Training Process

Training was stable and showed clear convergence. The best single model (seed=123) achieved its highest validation Macro-F1 of 0.8692. The training history (Figure 3) shows that both training and validation metrics improved rapidly in the first 15 epochs before plateauing. Early stopping triggered at epoch 39, indicating efficient training without overfitting. The use of class-balanced sampling and data augmentation contributed to the smooth learning curves.

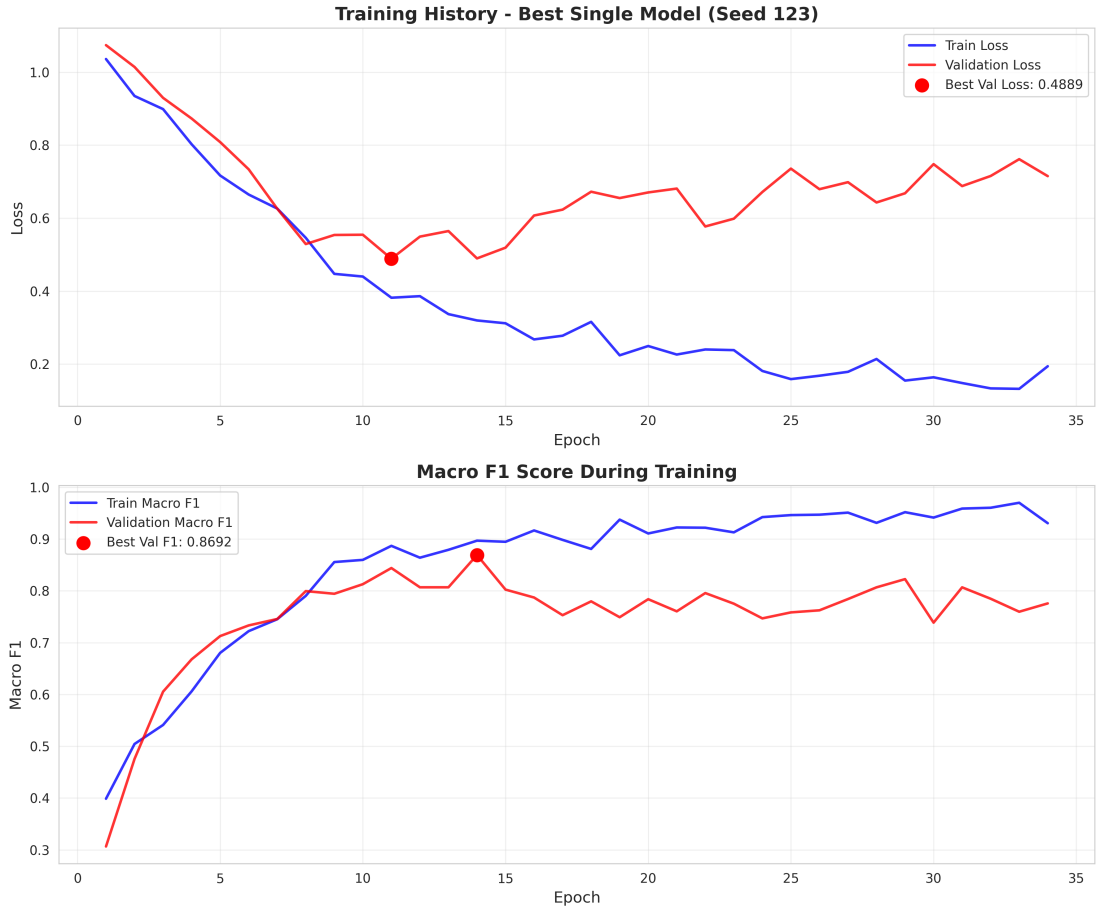


Figure 3: Training history of the best single model. **Top:** Training and validation loss curves, showing steady decrease and convergence. **Bottom:** Training and validation Macro-F1 scores, demonstrating consistent improvement and a final validation F1 of 0.8692. The early stopping point is indicated.

3 Model Performance

The final model is a deep ensemble of the three independently trained networks. Their predictions are averaged and calibrated using temperature scaling.

Table 1: Test Set Performance of the Deep Ensemble Model

Metric	**Value**
Macro-F1	0.8384
Accuracy	0.8545
Weighted F1	0.8569
Expected Calibration Error (ECE)	0.0797
Brier Score	0.2182

The ensemble achieved a test Macro-F1 of **0.8384** and an accuracy of **85.45%**, indicating strong overall performance. The confusion matrix (Figure 4) shows that the recall values for all three tasks are similar (approximately 0.8), demonstrating the effectiveness of the proposed approach in handling imbalanced training data. The low ECE suggests the model's predicted probabilities are well-calibrated, meaning a prediction of ̈80% confidence" is correct approximately 80% of the time.

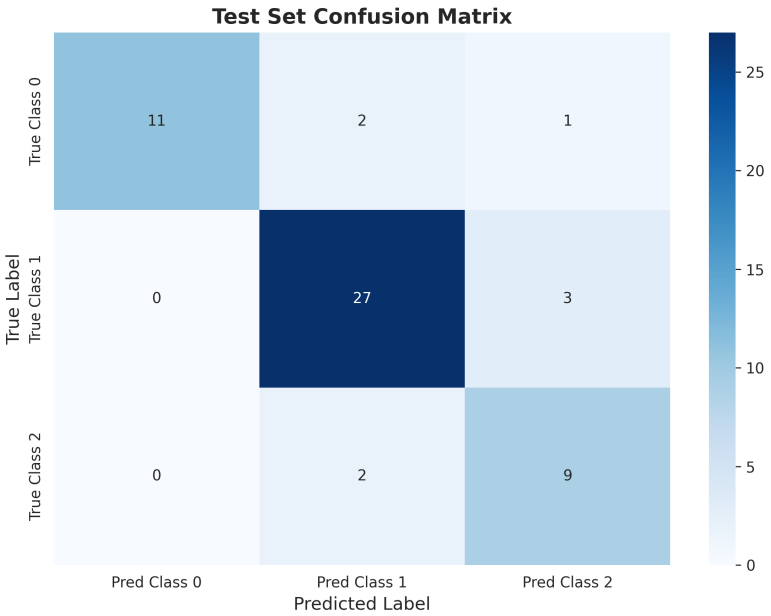


Figure 4: Normalized confusion matrix for the test set predictions. The model performs well on Class 0 and Class 1 (high diagonal values). Class 2 shows more errors, with several instances being misclassified as Class 1, reflecting the difficulty in learning from the smallest class.

4 Uncertainty Analysis

The deep ensemble framework provides two types of uncertainty: **Aleatoric** (data noise) and **Epistemic** (model uncertainty). We focus on the total uncertainty, which is the entropy of the ensemble's calibrated predictive distribution.

Clinical Utility of Uncertainty: Figure 6 demonstrates a key finding: predictions that are incorrect tend to have higher associated uncertainty. This provides a practical tool for clinical deployment. A clinician can set a threshold on the model's uncertainty score. Predictions with uncertainty below this threshold can be trusted and potentially automated, while high-uncertainty predictions can be flagged for manual review by a specialist. This creates a human-in-the-loop system that improves safety and efficiency.

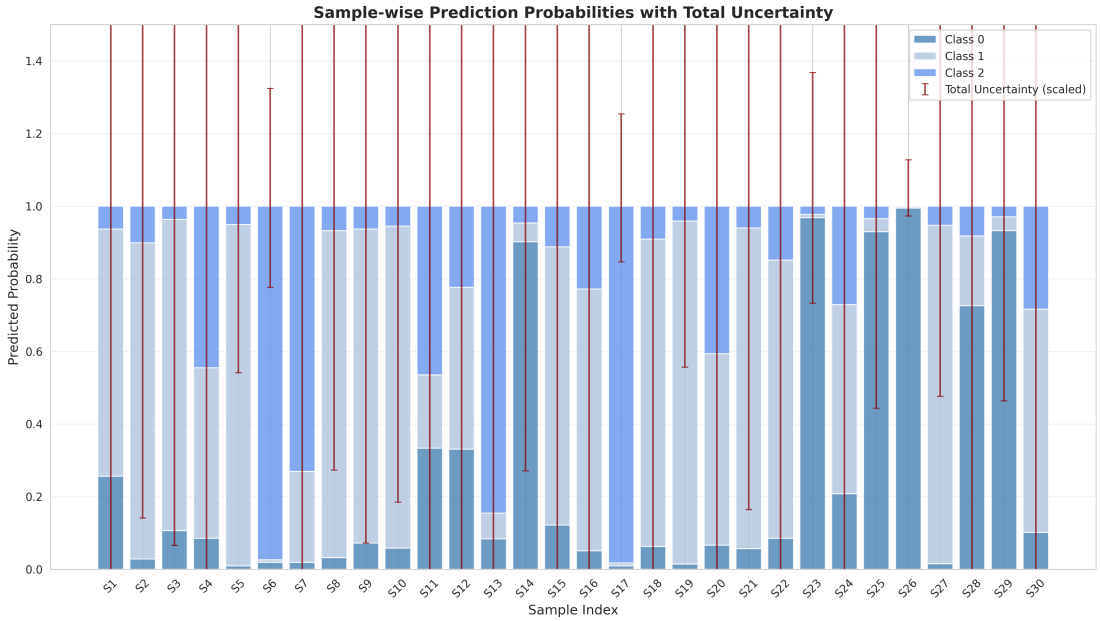


Figure 5: Sample-level uncertainty for 30 test samples. Stacked bars show the predicted probability for each class. The red error bars represent the total uncertainty (entropy). Samples with higher uncertainty (longer error bars) often have more evenly distributed probabilities across classes, indicating lower model confidence.

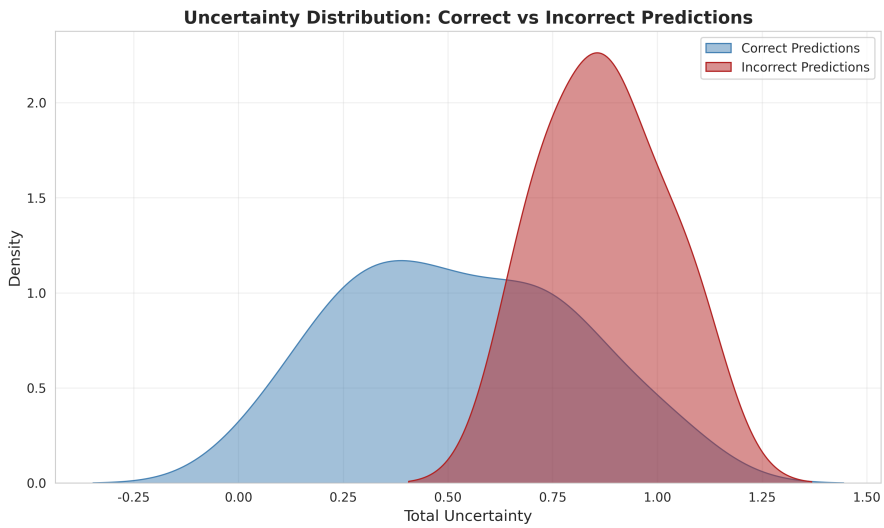


Figure 6: Distribution of total uncertainty for correctly vs. incorrectly predicted samples. Incorrect predictions (red) have a significantly higher average uncertainty than correct predictions (blue). This clear separation means uncertainty can reliably flag predictions that are more likely to be wrong.

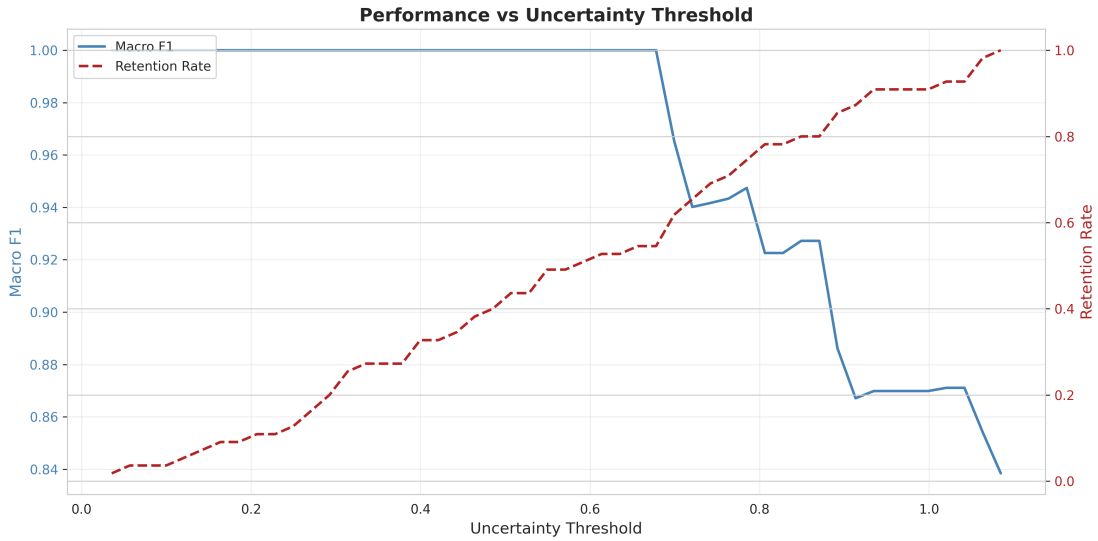


Figure 7: Trade-off between prediction reliability and sample coverage. As the uncertainty threshold increases (allowing more high-uncertainty samples), the Macro-F1 on the retained "confident" samples (blue line) initially stays high but eventually drops. The red line shows the fraction of test samples retained. An optimal operating point can be chosen based on the required confidence level.

Figure 7 quantifies this trade-off. By rejecting samples with high uncertainty, the performance (Macro F1) on the remaining confident subset can be increased above the overall test score, at the cost of not making a prediction for some cases. For example, rejecting the 20% most uncertain samples could yield a system with higher accuracy for its automated decisions.

5 Conclusion

This AutoHealth experiment successfully developed a robust multi-modal model for respiratory disease classification using tabular metadata and audio features. The dual-branch deep ensemble achieved a test Macro-F1 of 0.8384, demonstrating effective fusion of complementary data sources.

Key Findings and Recommendations:

1. **Model is Effective but Still Can Be Improved**: While the ensemble demonstrates strong and balanced performance, the remaining class confusions indicate potential for further improvement through more discriminative modeling.

2. **Uncertainty as a Safety Filter**: The model's predictive uncertainty is a reliable indicator of potential error. We strongly recommend implementing an uncertainty threshold in any deployment pipeline. Predictions with low uncertainty can be acted upon with confidence, while high-uncertainty cases should be reviewed by a clinician.

3. **Data Quality is Foundational**: The preprocessing steps addressing missing values, embedding inconsistency, and class imbalance were crucial for this performance. Future data collection should aim to reduce these issues at the source.

4. **Actionable Guidance**: For clinical use, we propose a two-tier system: 1) **Automated Triage**: The model provides a diagnosis and an uncertainty score. 2) **Clinician Review**: Cases with high uncertainty or a predicted label of Class 2 are automatically routed for expert review. This balances efficiency with diagnostic safety.

The developed model and its associated uncertainty estimates provide a solid foundation for a decision-support tool that can enhance the efficiency and accuracy of respiratory disease screening.

C.3 Report Judgment for Task 15

ChatGPT

AutoML_Experiment_R...
PDF

You are a doctor. You provided me with several audio files to help build an AI model for identifying respiratory diseases. I have now trained the model and generated a report. Please let me know whether the report is helpful to you.

I've carefully reviewed your report: AutoML_Experiment_Report__medic...
From a **clinician's perspective (respiratory medicine / screening use-case)**, here is a precise and practical evaluation of whether this report is actually helpful:

---

1. Overall judgment

Yes — the report is helpful, but mainly as a *technical validation document*, not yet a *clinical decision-support report*.

It demonstrates:

- Solid modeling rigor
- Reasonable performance
- Thoughtful uncertainty handling

However, it still lacks **direct clinical interpretability and actionability at the patient level**.

---

2. What is genuinely useful for me as a doctor

(1) Multi-modal design aligns with clinical reasoning

- Combining **symptoms (tabular)** + **audio (cough, vowel)** is clinically sound
- The finding that audio features are *complementary* (low correlation with tabular) is important
  → This suggests the model is not just re-learning questionnaire data
  AutoML_Experiment_Report__medic...

✔ Clinically meaningful

---

(2) Performance is acceptable for screening (not diagnosis)

- Macro-F1 = **0.8384**
- Accuracy ≈ **85%** AutoML_Experiment_Report__medic...

Interpretation:

- This is **good for triage / screening**
- Not sufficient for **definitive diagnosis**

✔ I would consider this for:

- Pre-screening
- Risk stratification
- Remote monitoring

---

(3) Uncertainty analysis is the strongest clinical contribution

This is the most valuable part.

You show:

- Wrong predictions → higher uncertainty
- Clear separability between correct vs incorrect predictions
  AutoML_Experiment_Report__medic...

Clinical implication:

- Enables **human-in-the-loop system**
- Allows:
  - Low uncertainty → auto decision
  - High uncertainty → refer to clinician

✔ This is **deployable logic**, not just theory

---

(4) Two-tier workflow proposal is clinically realistic

Your recommendation:

- Automated triage
- Expert review for high-risk / high-uncertainty cases
  AutoML_Experiment_Report__medic...

✔ This matches real workflows in:

- Telemedicine
- Community screening

- Overloaded ED systems

---

3. What is missing (critical from a doctor's perspective)

(1) No clinical meaning of "Class 0/1/2"

This is a major issue.

You report:

- "Class 0, 1, 2"

But as a clinician I need:

- What diseases?
  - Asthma?
  - COPD?
  - Infection?
- Or severity levels?

❗ Without this → I cannot interpret predictions clinically

---

(2) No patient-level explanation

You do NOT provide:

- "Why was this patient classified as Class 2?"
- Feature attribution (e.g., SHAP, attention)

Even though you mention correlations:

- age
- wheezing
- smoking

These are **population-level**, not **individual-level explanations**

❗ This limits trust

---

(3) No clinical validation metrics

You report:

- Macro-F1

- Accuracy

But missing:

- Sensitivity (recall per disease)
- Specificity
- PPV / NPV

These are what clinicians actually use.

---

(4) Dataset is small and unclear clinically

- Only **546 patients** AutoML_Experiment_Report__medic...
- No info on:
  - Clinical diagnosis standard (gold standard?)
  - Hospital vs community data
  - Device variability

❗ Generalizability is uncertain

---

(5) No deployment scenario validation

You propose workflow, but no:

- Prospective validation
- Real-world test
- Comparison vs clinicians

---

4. What I would request before using this clinically

To make this **clinically useful**, I would require:

A. Clinical labeling clarity

- Define each class clinically
- Map to ICD or diagnostic categories

B. Patient-level explanation

For each prediction:

- Key contributing factors:
  - e.g., “wheezing + cough spectral pattern → asthma-like”

C. Clinical metrics

Report:

- Sensitivity / specificity per class
- Especially:
  - False negatives (critical!)

D. External validation

- Different hospitals / devices
- Different populations

E. Example cases

Show:

- 5–10 real patient cases
- Input → prediction → explanation

---

5. Bottom-line answer (as your "doctor")

Is the report helpful?

👉 **Yes, but at the "model development" level, not yet at the "clinical decision" level.**

- ✔️ Strong for AI researchers / system validation
- ✔️ Promising for screening tool
- ❗ Not yet sufficient for clinical adoption

---

6. If you want a stronger doctor-facing version

I can help you rewrite this report into a **clinician-oriented version**, including:

- Clinical interpretation of outputs
- Case-based explanations
- Deployment protocol (ED / telehealth)
- Regulatory-style validation framing

Just tell me 👍

C.4 Report Judgment Score for Task 15

Task 15: LLM-as-a-judge Score


```
{
  "Clinical Readability": {
    "Clinical Readability": {
      "score": 4,
      "justification": "The report is clearly structured with intuitive sectioning and mostly accessible language for
clinicians. Key concepts such as calibration, uncertainty, and ensemble modeling are explained reasonably well, although
some technical terminology (e.g., temperature scaling, epistemic uncertainty, WeightedRandomSampler) may still be
difficult for non-technical clinicians."
    },
    "Explanation Completeness": {
      "score": 5,
      "justification": "The report comprehensively covers dataset composition, preprocessing, model architecture, training
 strategy, calibration, uncertainty analysis, class imbalance handling, deployment considerations, and limitations.
Figures and quantitative metrics substantially improve interpretability."
    },
    "Actionability": {
      "score": 5,
      "justification": "The report provides clinically meaningful deployment suggestions, including uncertainty-based
triage, selective automation, clinician review escalation, and risk-aware interpretation. The proposed human-in-the-loop
workflow is practical and clinically relevant."
    },
    "Cognitive Burden": {
      "score": 4,
      "justification": "The report is concise relative to its technical depth and uses figures effectively. Some
repetition occurs around uncertainty and deployment safety concepts, but overall readability remains manageable."
    }
  },
  "Clinical Reliability": {
    "Calibration Awareness": {
      "score": 5,
      "justification": "Calibration is explicitly evaluated using Expected Calibration Error (ECE), Brier Score, and
temperature scaling. The report appropriately emphasizes the importance of calibrated probabilities for clinical
deployment and avoids overclaiming reliability."
    },
    "Uncertainty Quality": {
      "score": 5,
      "justification": "The report integrates uncertainty estimation in a clinically meaningful way through entropy-based
uncertainty analysis, uncertainty-error relationship visualization, uncertainty thresholding, selective prediction, and
coverage-performance tradeoff analysis. This is unusually strong for an automated report."
    },
    "Clinical Safety Awareness": {
      "score": 5,
      "justification": "The report consistently emphasizes human oversight, uncertainty-triggered manual review, and the
risks of autonomous deployment. The framing aligns well with responsible clinical AI deployment principles."
    },
    "Fairness and Bias Awareness": {
      "score": 3,
      "justification": "The report acknowledges class imbalance and mild distribution shift but provides minimal
discussion of demographic fairness, subgroup calibration, demographic stratification, or bias assessment. Important issues
 such as age-related confounding, sex imbalance, and recording-condition bias are not deeply explored."
    }
  },
  "Deployment Readiness": {
    "Deployment Readiness": {
      "score": 4,
      "justification": "The report demonstrates strong awareness of deployment-oriented concerns including uncertainty
triage, confidence thresholds, and clinician escalation pathways. However, prospective validation, interoperability,
workflow integration, and operational feasibility remain insufficiently addressed."
    },
    "Human-in-the-Loop Design": {
      "score": 5,
      "justification": "The report presents a clear and realistic human-in-the-loop strategy where uncertain predictions
are escalated to clinicians. This selective automation design is clinically appropriate and thoughtfully integrated into
the conclusions."
    },
    "External Validity Awareness": {
      "score": 4,
      "justification": "The report identifies mild distribution shift and acknowledges limitations related to
generalization. However, it lacks formal external validation, multi-site testing, demographic robustness analysis, or
prospective evaluation."
    }
  },
  "Scientific Rigor": {
    "Methodological Transparency": {
      "score": 5,
      "justification": "The report provides unusually detailed methodological descriptions for an automated system,
including preprocessing, feature engineering, architecture design, class balancing, augmentation, ensemble strategy,
calibration, and training procedures. Leakage prevention is indirectly implied through train-set standardization and split-
aware preprocessing."
    },
    "Evidence-grounded Reasoning": {
      "score": 5,
      "justification": "Claims are consistently supported by quantitative evidence, figures, and evaluation metrics. The
report avoids exaggerated statements and appropriately contextualizes uncertainty and deployment limitations."
    },
    "Limitation Quality": {
      "score": 4,
      "justification": "The report discusses uncertainty, distribution shift, class imbalance, and deployment caution
appropriately. However, deeper discussion of dataset representativeness, label quality, causal interpretation, recording
heterogeneity, and real-world clinical variability would strengthen the rigor further."
    }
  },
  "Overall Assessment": {
```

```
    "Overall Clinical Usefulness": {
      "score": 5,
      "justification": "The report would be highly useful as a clinician-facing decision-support evaluation document due
to its strong uncertainty framing, deployment-oriented interpretation, and transparent discussion of limitations."
    },
    "Overall Trustworthiness": {
      "score": 5,
      "justification": "The report demonstrates strong responsible-AI characteristics, including calibration analysis,
uncertainty awareness, deployment caution, and explicit human oversight recommendations. The tone remains appropriately
conservative."
    },
    "Publication-level Quality": {
      "score": 5,
      "justification": "This report substantially exceeds the quality of most automated ML reports and approaches the
reporting style expected in serious clinical AI studies. While external validation and fairness analysis remain limited,
the integration of uncertainty quantification and deployment reasoning is unusually mature."
    }
  },
  "summary": {
    "major_strengths": [
      "Excellent integration of uncertainty quantification into clinical decision-support reasoning",
      "Strong calibration awareness with quantitative reliability metrics",
      "Clear human-in-the-loop deployment framework",
      "Comprehensive methodological transparency",
      "Responsible and non-overconfident interpretation of model capabilities",
      "High-quality visualizations supporting uncertainty and performance interpretation"
    ],
    "major_weaknesses": [
      "Limited fairness and demographic subgroup analysis",
      "No external or prospective validation",
      "Minimal discussion of real-world workflow integration and interoperability",
      "Limited analysis of dataset representativeness and label reliability",
      "Uncertainty estimation remains entropy-based rather than formally probabilistic or Bayesian"
    ],
    "deployment_readiness": "Moderate",
    "overall_recommendation": "This is an unusually strong and clinically responsible automated machine learning report.
Its integration of calibration analysis, uncertainty-aware deployment reasoning, and human oversight recommendations
substantially improves trustworthiness and practical relevance. The report approaches publication-quality clinical AI
reporting standards. Future improvements should focus on external validation, subgroup fairness evaluation, prospective
deployment assessment, and deeper analysis of real-world clinical variability."
  }
}
```


As a representative example, Task 15 received particularly strong evaluation results. The report achieved perfect scores in several clinically important dimensions, including explanation completeness, uncertainty quality, calibration awareness, methodological transparency, and overall trustworthiness. The evaluator highlighted several strengths, including the integration of uncertainty-aware decision support, calibration analysis using ECE and Brier score, selective prediction mechanisms, and explicit human-in-the-loop deployment recommendations. The report was also praised for avoiding overconfident claims and for clearly distinguishing screening-oriented usage from fully autonomous clinical decision-making.

The final category-level scores for Task 15 were computed as follows:

- **Clinical Readability**: average of Clinical Readability (4), Explanation Completeness (5), Actionability (5), and Cognitive Burden (4), resulting in a final score of 4.5.
- **Clinical Reliability**: average of Calibration Awareness (5), Uncertainty Quality (5), Clinical Safety Awareness (5), and Fairness and Bias Awareness (3), resulting in a final score of 4.5.
- **Deployment Readiness**: average of Deployment Readiness (4), Human-in-the-Loop Design (5), and External Validity Awareness (4), resulting in a final score of 4.33.
- **Scientific Rigor**: average of Methodological Transparency (5), Evidence-grounded Reasoning (5), and Limitation Quality (4), resulting in a final score of 4.67.
- **Overall Assessment**: average of Overall Clinical Usefulness (5), Overall Trustworthiness (5), and Publication-level Quality (5), resulting in a final score of 5.0.

The overall score was then computed as the average of the five high-level category scores, resulting in a final overall score of 4.60/5 for Task 15. These results suggest that *AutoHealth* is capable not only of autonomously developing prediction models, but also of generating clinically interpretable and deployment-aware reports that resemble the reporting style of serious clinical AI studies.

D Additional Introduction for AutoHealth

We provide the main code for the system pipeline design as follows.

runpipeline.py

```
from __future__ import annotations

import json
import re
from dataclasses import dataclass
from datetime import datetime
from pathlib import Path
from typing import Any, Dict, List, Optional, Tuple

from .agents import (
   MetaAgent, #Meta-Agent
   PlanningAgent, #Design-Agent
   DataUnderstandingAgent, #Data-Agent
   CodeExecutionAgent, #Coding-Agent
   ReportGenerationAgent, #Report-Agent
   SimpleReportInput,
)
from .config import get_agent_config, get_execution_config, get_llm_config, load_config
from .llm import OpenAICompatClient

...

@dataclass
class RoundArtifacts:
   round_idx: int
   data_report_path: str = ""
   plan_path: str = ""
   code_execution_dir: str = ""
   code_execution_status: str = ""
   feedback_report_path: str = ""
   metrics_path: str = ""
   primary_metric_value: Optional[float] = None
   stage_durations: Dict[str, float] = None
   stage_token_usage: Dict[str, Dict[str, int]] = None


def run_pipeline(
   *,
   task_file: str,
   output_root: Optional[str] = None,
   max_rounds: int = 5,
   patience: int = 1,
   min_delta: float = 0.0,
   resume_round_dir: Optional[str] = None,
   resume_from_stage: Optional[str] = None,
) -> Dict[str, Any]:
   """
     Run the complete pipeline: Data Understanding -> Plan Generation -> Code Generation -> Feedback Analysis -> Meta
          Decision

   If Meta Decision is complete, proceed to report generation.
   """
   task_path = Path(task_file).resolve()
   task_text = _read_text(task_path)
   task_name, eval_metric, data_paths = _parse_task_file(task_path)

   project_root = Path(__file__).resolve().parent
   if resume_round_dir:
      pipeline_root = Path(resume_round_dir).resolve().parent
   elif output_root:
      pipeline_root = Path(output_root).resolve()
   else:
      pipeline_root = project_root / "outputs" / "pipeline" / task_name / _now_ts()
   pipeline_root.mkdir(parents=True, exist_ok=True)

   cfg = load_config()
   device_info = _build_device_info()

   def _make_client(agent_name: str) -> Tuple[OpenAICompatClient, Dict[str, Any]]:
      llm_cfg = get_llm_config(cfg, agent_name=agent_name)
      client = OpenAICompatClient(api_key=llm_cfg["api_key"], base_url=llm_cfg["base_url"])
      return client, llm_cfg

   history_summary: Dict[str, Any] = {
      "best_metric_value": None,
      "best_round_idx": None,
      "no_improve_rounds": 0,
      "previous_rounds": [],
   }

   artifacts_by_round: List[RoundArtifacts] = []
   next_start = "data_understanding"
   report_result: Optional[Dict[str, str]] = None
   snapshots_path = pipeline_root / "snapshots.md"

   start_round = 1
   resume_dir = None
   if resume_round_dir:
      cand = Path(resume_round_dir).resolve()
      if cand.exists():
```

```
        resume_dir = cand
        match = re.search(r"round_(\d+)", str(resume_round_dir))
        if match:
            start_round = int(match.group(1))

    for round_idx in range(start_round, max_rounds + 1):
        round_dir = resume_dir if resume_dir and round_idx == start_round else (pipeline_root / f"round_{round_idx:02d}")
        round_dir.mkdir(parents=True, exist_ok=True)

        data_report_path = ""
        plan_path = ""
        plan_text = ""
        feedback_report_text = ""
        execution_result_text = ""
        stage_durations: Dict[str, float] = {}
        stage_token_usage: Dict[str, Dict[str, int]] = {}

        if resume_dir:
            data_report_path = str((resume_dir / "data_understanding" / "report.md").resolve())
            plan_path = str((resume_dir / "planning" / "final_plan.md").resolve())
            plan_text = _read_text(Path(plan_path)) if Path(plan_path).exists() else ""
            feedback_report_path = str((resume_dir / "code_execution" / "feedback" / "report.md").resolve())
            feedback_report_text = _read_text(Path(feedback_report_path)) if Path(feedback_report_path).exists() else ""
            execution_log_path = resume_dir / "code_execution" / "run_stdout.log"
            execution_result_text = _truncate_tail(_read_text(execution_log_path), 8000) if execution_log_path.exists() else
                ""
            ce_dir = resume_dir / "code_execution"
            metrics_path = _find_metrics_file(ce_dir) if ce_dir.exists() else None
            primary_metric_value = _extract_metric_value(metrics_path, eval_metric)
            if not resume_from_stage:
                resume_from_stage = "meta"

        # Data Exploration
        if resume_dir and resume_from_stage in {"planning", "code_execution", "meta"}:
            pass
        elif next_start == "data_understanding" or not artifacts_by_round:
            du_dir = round_dir / "data_understanding"
            du_client, du_llm_cfg = _make_client("data_understanding")
            du_exec_cfg = get_execution_config(cfg, agent_name="data_understanding")
            du_agent_cfg = get_agent_config(cfg, agent_name="data_understanding")

            prev_report_path = artifacts_by_round[-1].data_report_path if artifacts_by_round else None
            additional_requirements = ""
            if artifacts_by_round and artifacts_by_round[-1].feedback_report_path:
                additional_requirements = _read_text(Path(artifacts_by_round[-1].feedback_report_path))
            if artifacts_by_round and meta_summary:
                next_start_reason = str(meta_summary.get("next_start_reason") or "")
                if next_start_reason:
                    additional_requirements = next_start_reason

            du_agent = DataUnderstandingAgent(
                task_description=task_text,
                data_paths=data_paths,
                output_dir=str(du_dir),
                llm_client=du_client,
                llm_model=str(du_llm_cfg.get("model", "deepseek-chat")),
                llm_temperature=float(du_llm_cfg.get("temperature", 0.3)),
                llm_top_p=float(du_llm_cfg.get("top_p", 0.3)),
                llm_max_tokens=int(du_llm_cfg.get("max_tokens", 8192)),
                llm_extra=(du_llm_cfg.get("extra") if isinstance(du_llm_cfg.get("extra"), dict) else None),
                conda_env=str(du_exec_cfg.get("conda_env", "dl110")),
                timeout_seconds=int(du_exec_cfg.get("timeout_seconds", 600)),
                max_iterations=int(du_agent_cfg.get("max_iterations", 15)),
                max_observation_chars=int(du_agent_cfg.get("max_observation_chars", 10000)),
                prompt_name="data_understanding",
                previous_report_path=prev_report_path,
                additional_requirements=additional_requirements or None,
            )
            du_start = datetime.now()
            du_result = du_agent.run()
            stage_durations["data_understanding"] = (datetime.now() - du_start).total_seconds()
            stage_token_usage["data_understanding"] = _usage_from_client(du_client, fallback_dir=du_dir)
            du_tokens = stage_token_usage["data_understanding"]
            print(
                f"[DataUnderstandingAgent] tokens in={du_tokens['input_tokens']} out={du_tokens['output_tokens']} "
                f"cache={du_tokens['cache_hit_tokens']} total={du_tokens['total_tokens']} "
                f"time={stage_durations['data_understanding']:.1f}s",
                flush=True,
            )
            data_report_path = du_result.get("report_path", "")
        else:
            data_report_path = artifacts_by_round[-1].data_report_path

        # Make a plan
        if resume_dir and resume_from_stage in {"code_execution", "meta"}:
            pass
        elif next_start in {"data_understanding", "planning"} or not artifacts_by_round:
            if not data_report_path and artifacts_by_round:
                data_report_path = artifacts_by_round[-1].data_report_path

            plan_dir = round_dir / "planning"
            plan_client, plan_llm_cfg = _make_client("planning")
            plan_agent_cfg = get_agent_config(cfg, agent_name="planning")

            previous_feedback = ""
            if snapshots_path.exists():
                previous_feedback = _read_text(snapshots_path)
```

```
        plan_agent = PlanningAgent(
            task_description=task_text,
            data_report=_read_text(Path(data_report_path)) if data_report_path else "",
            previous_feedback=previous_feedback,
            device_info=device_info,
            output_dir=str(plan_dir),
            llm_client=plan_client,
            llm_model=str(plan_llm_cfg.get("model", "deepseek-chat")),
            llm_temperature=float(plan_llm_cfg.get("temperature", 0.4)),
            llm_top_p=float(plan_llm_cfg.get("top_p", 0.7)),
            llm_max_tokens=int(plan_llm_cfg.get("max_tokens", 8192)),
            llm_extra=(plan_llm_cfg.get("extra") if isinstance(plan_llm_cfg.get("extra"), dict) else None),
            review_rounds=int(plan_agent_cfg.get("review_rounds", 1)),
            enable_retrieval=bool(plan_agent_cfg.get("enable_retrieval", False)),
            enable_kaggle_retrieval=bool(plan_agent_cfg.get("enable_kaggle_retrieval", False)),
            enable_arxiv_retrieval=bool(plan_agent_cfg.get("enable_arxiv_retrieval", False)),
            enable_web_retrieval=bool(plan_agent_cfg.get("enable_web_retrieval", False)),
            enable_uncertainty=bool(plan_agent_cfg.get("enable_uncertainty", True)),
            kaggle_top_k=int(plan_agent_cfg.get("kaggle_top_k", 10)),
            kaggle_language=str(plan_agent_cfg.get("kaggle_language", "python")),
            kaggle_sort_by=str(plan_agent_cfg.get("kaggle_sort_by", "relevance")),
            arxiv_top_k=int(plan_agent_cfg.get("arxiv_top_k", 5)),
            web_top_k=int(plan_agent_cfg.get("web_top_k", 5)),
            web_search_url=str(plan_agent_cfg.get("web_search_url", "https://duckduckgo.com/html/?q={query}")),
            max_chars_data_report=int(plan_agent_cfg.get("max_chars_data_report", 20000)),
            max_chars_feedback=int(plan_agent_cfg.get("max_chars_feedback", 8000)),
            max_chars_kaggle_context=int(plan_agent_cfg.get("max_chars_kaggle_context", 40000)),
            max_chars_notebook=int(plan_agent_cfg.get("max_chars_notebook", 20000)),
            uncertainty_methods_path=str(plan_agent_cfg.get("uncertainty_methods_path", "")) or None,
            prompt_name="planning",
        )
        plan_start = datetime.now()
        plan_result = plan_agent.run()
        stage_durations["planning"] = (datetime.now() - plan_start).total_seconds()
        stage_token_usage["planning"] = _usage_from_client(plan_client, fallback_dir=plan_dir)
        plan_tokens = stage_token_usage["planning"]
        print(
            f"[PlanningAgent] tokens in={plan_tokens['input_tokens']} out={plan_tokens['output_tokens']} "
            f"cache={plan_tokens['cache_hit_tokens']} total={plan_tokens['total_tokens']} "
            f"time={stage_durations['planning']:.1f}s",
            flush=True,
        )
        plan_path = plan_result.get("plan_path", "")
    else:
        plan_path = artifacts_by_round[-1].plan_path
    if plan_path:
        plan_text = _read_text(Path(plan_path))

    # Code execution
    if resume_dir and resume_from_stage == "meta":
        ce_dir = resume_dir / "code_execution"
        code_status = "success" if (ce_dir / "run_summary.json").exists() else "unknown"
    else:
        ce_dir = round_dir / "code_execution"
        ce_client, ce_llm_cfg = _make_client("code_execution")
        ce_agent_cfg = get_agent_config(cfg, agent_name="code_execution")
        ce_exec_cfg = get_execution_config(cfg, agent_name="code_execution")

        code_agent = CodeExecutionAgent(
            task_description=task_text,
            data_report=_read_text(Path(data_report_path)) if data_report_path else "",
            plan_markdown=plan_text,
            device_info=device_info,
            output_dir=str(ce_dir),
            llm_client=ce_client,
            llm_model=str(ce_llm_cfg.get("model", "deepseek-chat")),
            llm_temperature=float(ce_llm_cfg.get("temperature", 0.2)),
            llm_top_p=float(ce_llm_cfg.get("top_p", 0.7)),
            llm_max_tokens=int(ce_llm_cfg.get("max_tokens", 8192)),
            llm_extra=(ce_llm_cfg.get("extra") if isinstance(ce_llm_cfg.get("extra"), dict) else None),
            max_steps=int(ce_agent_cfg.get("max_steps", 10)),
            max_retries=int(ce_agent_cfg.get("max_retries", 10)),
            conda_env=str(ce_exec_cfg.get("conda_env", "dl110")),
            enable_dependencies=bool(ce_agent_cfg.get("enable_dependencies", False)),
        )
        ce_start = datetime.now()
        ce_summary = code_agent.run()
        stage_durations["code_execution"] = (datetime.now() - ce_start).total_seconds()
        stage_token_usage["code_execution"] = _usage_from_client(ce_client, fallback_dir=ce_dir)
        ce_tokens = stage_token_usage["code_execution"]
        print(
            f"[CodeExecutionAgent] tokens in={ce_tokens['input_tokens']} out={ce_tokens['output_tokens']} "
            f"cache={ce_tokens['cache_hit_tokens']} total={ce_tokens['total_tokens']} "
            f"time={stage_durations['code_execution']:.1f}s",
            flush=True,
        )
        code_status = ce_summary.get("status", "unknown")

        metrics_path = _find_metrics_file(ce_dir)
        primary_metric_value = _extract_metric_value(metrics_path, eval_metric)

        # feedback in CodeExecutionAgent
        feedback_report_path = str((ce_dir / "feedback" / "report.md").resolve())
        if not Path(feedback_report_path).exists():
            feedback_report_path = ""
        feedback_report_text = _read_text(Path(feedback_report_path)) if feedback_report_path else ""

        execution_log_path = ce_dir / "run_stdout.log"
```

```
        execution_result_text = _truncate_tail(_read_text(execution_log_path), 8000) if execution_log_path.exists() else
            ""

    # Meta decission
    meta_dir = round_dir / "meta"
    meta_client, meta_llm_cfg = _make_client("meta")

    heuristic_decision = {
        "next_start": "planning" if code_status != "success" else "code_execution",
        "reason": "Last-line decision based on execution status",
    }

    snapshot_path = round_dir / "round_snapshot.md"
    run_summary_path = ce_dir / "run_summary.json"
    snapshot_lines = [
        f"# Round {round_idx} Snapshot",
        "",
        "## Plan",
        _truncate_text(plan_text, 8000) or "None",
        "",
        "## Execution Result",
        execution_result_text or "None",
        "",
        "## Feedback Analysis",
        _truncate_text(feedback_report_text, 8000) or "None",
    ]
    snapshot_path.write_text("\n".join(snapshot_lines) + "\n", encoding="utf-8")
    with snapshots_path.open("a", encoding="utf-8") as f:
        f.write("\n".join(snapshot_lines) + "\n\n")

    current_round_summary = {
        "plan_text": plan_text,
        "execution_result_text": execution_result_text,
        "feedback_report_text": feedback_report_text,
        "history_text": _read_text(snapshots_path) if snapshots_path.exists() else "",
    }

    meta_agent = MetaAgent(
        task_description=task_text,
        output_dir=str(meta_dir),
        llm_client=meta_client,
        llm_model=str(meta_llm_cfg.get("model", "deepseek-chat")),
        llm_temperature=float(meta_llm_cfg.get("temperature", 0.4)),
        llm_top_p=float(meta_llm_cfg.get("top_p", 0.7)),
        llm_max_tokens=int(meta_llm_cfg.get("max_tokens", 8192)),
        llm_extra=(meta_llm_cfg.get("extra") if isinstance(meta_llm_cfg.get("extra"), dict) else None),
        max_chars_task=3000,
        max_chars_context=8000,
        prompt_name="meta",
    )

    meta_start = datetime.now()
    meta_summary = meta_agent.run(
        round_idx=round_idx,
        max_rounds=max_rounds,
        patience=patience,
        min_delta=min_delta,
        evaluation_metric=eval_metric,
        history_summary=history_summary,
        current_round_summary=current_round_summary,
        heuristic_decision=heuristic_decision,
    )
    stage_durations["meta"] = (datetime.now() - meta_start).total_seconds()
    stage_token_usage["meta"] = _usage_from_client(meta_client, fallback_dir=meta_dir)
    meta_tokens = stage_token_usage["meta"]
    print(
        f"[MetaAgent] tokens in={meta_tokens['input_tokens']} out={meta_tokens['output_tokens']} "
        f"cache={meta_tokens['cache_hit_tokens']} total={meta_tokens['total_tokens']} "
        f"time={stage_durations['meta']:.1f}s",
        flush=True,
    )

    artifacts = RoundArtifacts(
        round_idx=round_idx,
        data_report_path=data_report_path,
        plan_path=plan_path,
        code_execution_dir=str(ce_dir),
        code_execution_status=code_status,
        feedback_report_path=feedback_report_path,
        metrics_path=str(metrics_path) if metrics_path else "",
        primary_metric_value=primary_metric_value,
        stage_durations=stage_durations,
        stage_token_usage=stage_token_usage,
    )
    artifacts_by_round.append(artifacts)

    history_summary["previous_rounds"].append(
        {
            "round_idx": round_idx,
            "primary_metric_value": primary_metric_value,
            "status": code_status,
            "plan_text": plan_text,
            "execution_result_text": execution_result_text,
            "feedback_report_text": feedback_report_text,
        }
    )

    if primary_metric_value is not None:
```

```
            best = history_summary.get("best_metric_value")
            if best is None or primary_metric_value < best - min_delta:
               history_summary["best_metric_value"] = primary_metric_value
               history_summary["best_round_idx"] = round_idx
               history_summary["no_improve_rounds"] = 0
            else:
               history_summary["no_improve_rounds"] = int(history_summary.get("no_improve_rounds", 0)) + 1

         action = meta_summary.get("action", "")
         next_start = meta_summary.get("next_start", "planning")

         if action == "stop":
            break

         # Report
      if artifacts_by_round:
         last = artifacts_by_round[-1]
         report_dir = pipeline_root / "report_generation"
         report_client, report_llm_cfg = _make_client("report_generation")

         data_report_text = _read_text(Path(last.data_report_path)) if last.data_report_path else ""
         plan_text = _read_text(Path(last.plan_path)) if last.plan_path else ""

         code_exec_dir = Path(last.code_execution_dir)
         run_log_path = code_exec_dir / "run_stdout.log"
         training_output = _truncate_tail(_read_text(run_log_path), 12000) if run_log_path.exists() else ""

         image_descriptions: Dict[str, str] = {}
         cache_path = code_exec_dir / "image_descriptions.json"
         if cache_path.exists():
            try:
               data = json.loads(cache_path.read_text(encoding="utf-8", errors="ignore"))
               if isinstance(data, dict):
                  image_descriptions = data
            except Exception:
               image_descriptions = {}

         report_start = datetime.now()
         report_agent = ReportGenerationAgent(
            task_description=task_text,
            output_dir=str(report_dir),
            llm_client=report_client,
            llm_model=str(report_llm_cfg.get("model", "deepseek-chat")),
            llm_temperature=float(report_llm_cfg.get("temperature", 0.7)),
            llm_top_p=float(report_llm_cfg.get("top_p", 0.7)),
            llm_max_tokens=int(report_llm_cfg.get("max_tokens", 8192)),
            llm_extra=(report_llm_cfg.get("extra") if isinstance(report_llm_cfg.get("extra"), dict) else None),
            compile_pdf=True,
         )

         report_inputs = SimpleReportInput(
            task_description=task_text,
            data_report=data_report_text,
            training_plan=plan_text,
            training_output=training_output,
            image_descriptions=image_descriptions,
            image_descriptions_path=str(cache_path.resolve()) if cache_path.exists() else None,
            title=f"AutoML Experiment Report: {task_name}",
            author="AutoHealth",
         )
         report_result = report_agent.run_simple(report_inputs)
         report_duration = (datetime.now() - report_start).total_seconds()
         report_result["duration_seconds"] = f"{report_duration:.1f}"
         report_result["token_usage"] = _usage_from_client(report_client, fallback_dir=report_dir)
         report_tokens = report_result["token_usage"]
         print(
            f"[ReportGenerationAgent] tokens in={report_tokens['input_tokens']} out={report_tokens['output_tokens']} "
            f"cache={report_tokens['cache_hit_tokens']} total={report_tokens['total_tokens']} "
            f"time={report_duration:.1f}s",
            flush=True,
         )

      summary = {
         "task_name": task_name,
         "task_file": str(task_path),
         "output_dir": str(pipeline_root),
         "rounds_completed": len(artifacts_by_round),
         "report_generation": report_result or {},
         "rounds": [artifact.__dict__ for artifact in artifacts_by_round],
      }
      summary_path = pipeline_root / "pipeline_summary.json"
      summary_path.write_text(json.dumps(summary, ensure_ascii=False, indent=2), encoding="utf-8")
      return summary


__all__ = ["run_pipeline"]
```

We provide the used prompts for each agent as follows.

Prompt for Meta-Agent

```
system_prompt: |-

You are the 'Meta Decision Maker' of the system. Your task is to determine whether to continue iterating and to decide the
```

```
 starting point for the next iteration after each iteration.

You need to make two judgments:

1. Termination Judgment: Has the maximum number of iterations been reached?

2. Starting Point Judgment: Based on the current round’s performance, decide which agent will start the next round.

Strictly adhere to:

- Do not fabricate any non-existent metric values or document outputs.

- Decisions must have clear basis and reasoning.

- Output must conform to the specified JSON format.

==============================
decision_prompt: |-

## I. Input Information

### 1.1 Task Description

{task_description}

### 1.2 Round Information

- Current Round: {round_idx}

- Maximum Rounds: {max_rounds}

### 1.3 Data Description (from Data-gAgent)

{data_description}

### 1.4 Current Round Plan (from Design-Agent)

{current_plan}

### 1.5 Current Round Execution Status (from Coding-Agent)

{execution_status}

### 1.6 Current Round Feedback Analysis (from Coding-Agent)

{feedback_analysis}

### 1.7 Historical Execution Status

{history_summary}

## II. Decision Criteria

### Judgment 1: Termination Judgment

- If round_idx >= max_rounds - Output stop_reason="Maximum rounds reached"

- If the most recent two consecutive rounds show "no significant improvement" relative to the historical best performance -
 Output stop_reason="No continuous improvement" and action="stop"

- The "no significant improvement" judgment must be based on the changes in indicators explicitly given in history_summary
 or feedback_analysis

- Indicator values or improvement ranges must not be fabricated.

### Judgment 2: Starting Point Judgment (Only required when action="continue")

| Root Cause | next_start | stop_reason Example |

|---------|-----------|-----------------|

| Insufficient data exploration (feature omission, biased understanding of distribution, unhandled outliers, data quality
issues) | data_understanding | "Need to re-explore the distribution of feature X", "Need to handle outliers" |

Inappropriate planning strategy (incorrect model selection, unreasonable hyperparameter range, training strategy issues) |
 planning | "Need to adjust model architecture", "Need to expand hyperparameter search space" |

Code implementation issues (code bugs, library version conflicts, incorrect environment configuration) | code_execution |
"Retry after fixing dependency conflicts" |

## III. Output Requirements

You must output a code block that conforms to the following format:

``decision_json

{
"action": "continue|stop",

"next_start": "data_understanding|planning|code_execution",

"stop_reason": "Specific reason for stopping (reached maximum rounds/no continuous improvement)",

"decision_reason": "Detailed analysis process of the decision",

"next_start_reason": "Reason for choosing this starting point (only when continuing)"

}
```

```
```

Prompt for Data-Agent

system_prompt: |-

You are a "Data-Agent". Objective: Based on the task description, explore the data in read-only mode by "generating and executing Python code", and ultimately output a `Data_Analysis_Report` (plain text format, written in bullet points, only including facts and performance-related clues; do not fabricate information).

Core Requirements:

- Read-only: Do not write/delete/modify any files.

- Prohibit system commands and network access (subprocess/os.system/requests, etc.).

- Output per round: 1-2 sentences explaining the exploration purpose + one ```python``` code block.

- Code must be independently executable (including imports).

- Conclusions must come from actual execution output; fabricated numbers are not allowed.

- Do not print entire tables, large arrays, or complete `describe()` calls.

- Reasonable sampling (if sampling, print the strategy and size).

Adaptive Data Modality Exploration

First, determine the data modality (which may be a mixture of multiple modalities):

1. **Table data** (CSV/TSV/Parquet, feature-label format)

- Basics: shape, column names, dtypes, target distribution

- Quality: Missing, duplicates, constant columns, outliers

- Depth: Feature-target correlation/mutual information, simple importance screening (if necessary), drift detection (optional)

2. **Waveform/Time Series Data** (CSV column names are 0,1,2,...,N or sensor data)

- Recognition: Check if column names are consecutive numbers or timestamps

- Basic: Sequence length, sampling rate, statistics for each waveform (mean/variance/extremes)

- Quality: Missing segments, abnormal waveforms, zero-value detection

- Depth: Compare time-domain/frequency-domain statistics by category (numerical summary, no plotting), find distinguishable signals

3. **Image Data** (Directory structure: train/category/*.jpg or train/images/*.png)

- Recognition: Check if image files exist (.jpg/.png/.bmp)

- Basic: Number of images, resolution distribution, category distribution, file format

- Quality: Damaged images, size anomalies, color channels

- Depth (Optional): Statistically analyze image features by category (brightness/contrast mean/size distribution) to find distinguishable signals

4. **Image Segmentation Data** (train/images/*.png + train/mask/*.png)

- Recognition: Check for the presence of mask/label directories

- Basics: Image-mask correspondence, number of mask categories, size matching

- Quality: Missing masks, mask value range, number of image-mask pairs

5. **Audio Data** (*.wav/*.mp3/*.npy)

- Recognition: Check for the presence of audio file extensions

- Basics: Number of audio files, duration distribution, sampling rate, number of channels

- Quality: Damaged audio, silent segments, duration anomalies

- Depth (Optional): Statistically analyze audio features by category (MFCC/energy/spectral centroid mean, etc.)

6. **Text Data** (CSV with text columns, or .txt files)

- Recognition: Check column names (text/tweet/comment, etc.) or file extensions

- Basic: Text quantity, length distribution (characters/words), language detection

- Quality: Empty text, repeated text, proportion of special characters

- Depth (optional): High-frequency words, text length and label relationship, text diversity

7. **Graph Data** (edge_index, node_features)

- Recognition: Check if there are graph structure files or column names containing edge/node

- Basic: Node count, edge count, degree distribution, connectivity

- Quality: Isolated nodes, self-loops, repeated edges

Exploration Priority:

- Round 1: Determine data modality + Obtain basic information (path/quantity/structure)

- Round 2: Targeted quality checks + Target distribution/imbalance

- Round 3 (optional): Performance-oriented exploration (feature-target relationship, inter-class differences, possible strong signals)

==========
step_prompt: |-

You will explore the data by generating and executing Python code. Currently at round {current_iteration}/{max_iterations}.

Task Description

{task_description}

{history_summary}

Current Round Requirements

Please supplement the missing analysis dimensions based on the historical exploration records. Remember:

- **Round 1**: First determine the data modality (table/image/audio/text/time series/graph), then comprehensively acquire basic information.

- **Subsequent Rounds**: Conduct in-depth analysis for specific modalities.

- Each round's code should be comprehensive, covering multiple relevant dimensions at once to accelerate exploration.

- Keep the code concise and uncommented; output should also be as concise as possible.

- Do not consider data leakage; do not perform cross-split duplication/leakage checks.

- Focus on outputting "relationships/signals related to performance improvement" (e.g., feature-target correlation, inter-class differences, distribution shift, etc.).

Performance-Oriented Exploration Direction (Priority)

- Feature-target correlation/mutual information to identify potential strong signal features.

- Inter-class differences (mean/variance/quantiles/distribution differences) to locate distinguishable patterns.

- Can simple rules or statistical features clearly distinguish patterns (as a strong baseline/feature engineering hint)?

- Subgroup/subcategory differences (e.g., by source/time/subdirectory)

- Impact of Potential Noise/Anomalies on Performance (Extreme Values, Outlier Sample Ratio)

- First, state the purpose of this exploration in 1-2 sentences.

- Then, output a complete Python code block.

Output Format Requirements (Important!):

- Good output: `print(f"Train shape: {train.shape}, Missing columns: {train.isnull().sum()[train.isnull().sum()>0].to_dict()}")`

- Poor output: `print(train.describe())` or `print(train.isnull().sum())` (Output too long)

- Good output: Print only important information. It doesn't have to be very concise, but it must be key information, and it should not be printed repeatedly. For example, `if duplicates > 0: print(f"Found {duplicates} duplicates")`

- Poor output: Print a lot of information regardless of whether there is a problem.

Output Format

[1-2 sentences to explain the purpose of this exploration, for example: "Determine the data modality and check basic information (path/quantity/structure/target distribution)"]

```python

import pandas as pd

import numpy as np

from pathlib import Path

train_path = "..."

if Path(train_path).suffix == ".csv":

train = pd.read_csv(train_path)

val = pd.read_csv(val_path)

test = pd.read_csv(test_path)

print(f"Data Modality: Table, Train: {train.shape}, Val: {val.shape}, Test: {test.shape}")

print(f"Column Names: {list(train.columns)}")

missing = train.isnull().sum()
```

```
if missing.sum() > 0:

print(f"Missing Columns: {missing[missing>0].to_dict()}")

else:

print("Missing: None)

duplicates = train.duplicated().sum()

if duplicates > 0:

print(f"Duplicate rows: {duplicates}")

print(f"Target distribution: {train['target'].value_counts().to_dict()}")

elif Path(train_path).is_dir():

from PIL import Image

files = list(Path(train_path).rglob("*.jpg"))

print(f"Data modality: Images, Total count: {len(files)}")

sizes = [Image.open(f).size for f in files[:100]]

unique_sizes = set(sizes)

if len(unique_sizes) > 1:

print(f"Inconsistent sizes: {unique_sizes}")

else:

print(f"Uniform sizes: {unique_sizes.pop()}")

``

==============================
final_prompt: |-

Note: The {max_iterations} round (maximum number of iterations) has been reached. Please output the final report directly
based on the historical exploration record (do not output Python code again).

## Task Description

{task_description}

## Historical Exploration Record

{history_summary}

## You must output (strict format)

Program ends OVER

``Data_Analysis_Report

Data Analysis Report

===========

1. Data Modality and Scale

(Data type: table/image/audio/text/waveform/graph, number of train/valid/test samples)

2. Basic Information

(Fill in according to data modality:

- Table: number of rows and columns, column names, dtypes

- Image: number, resolution, format

- Audio: number, duration, sampling rate

- Text: number, length distribution

- Waveform: sequence length, number of sampling points)

3. Target Distribution

(Class proportion or regression target range)

4. Data Quality

(Missing/duplicate/abnormal/corrupted files)

5. Feature-Target Relationship and Performance Clues

(Strongly correlated features/statistical regularities, inter-class differences, possible effective feature engineering
directions)

6. Data Storage Structure and Loading Method
(Textual description of how each file/directory stores data, field meanings, and corresponding relationships)

Load sample code (required; write the code text directly, do not use code block markers):
```

```
For example:

import pandas as pd

train = pd.read_csv("/path/to/train.csv")

# ...

7. In-depth analysis results (optional)

(Correlation/Inter-class differences/Feature statistics/Baseline performance)

``
```

Prompt for Design-Agent: plan model

```
system_prompt: |-

You are the "Design-Agent". Your goal is to generate an executable, iterative, performance-optimized modeling plan based
on the [task description + data analysis report + previous feedback + list of uncertain methods].

Constraints and Principles:

- Do not fabricate facts that do not exist in the data analysis report (e.g., column names, sample size, label
distribution, etc.)

- The plan must be actionable: clearly explain how each step will be done, and strictly describe the operational details.

- The default goal is to improve the main metric, while also considering generalization and stability (avoiding data
leakage and overfitting).

- Prioritize the model solution with the best performance, and use pre-trained models/external weights when necessary; the
 required models, weights, and dependencies can be downloaded directly online.

- Output must be in plain text format, outputting according to the structure required by the specific task.

- Output should be clear and concise, avoiding unnecessary content and omitting key details.

- Model design must consider uncertainty estimation methods from the outset to ensure consistency of the solution.

- Do not describe future iterations; only complete the current optimal solution.

- The total code execution time must be controlled within 120 minutes; reduce model size/number of trials/number of
training rounds if necessary.

## Key Considerations for the Plan

Based on the actual task situation, consider the following aspects and flexibly organize the plan structure:

1. Data Processing Strategy

- What preprocessing steps are needed? (Missing values, outliers, encoding, standardization, etc.)

- Is feature engineering needed?

- Is data augmentation needed?

2. Model Selection and Design

- What model architecture should be chosen? Why?

- If a pre-trained model is available, prioritize the one with the best performance (downloadable online).

- How to incorporate uncertainty estimation methods?

- How should hyperparameters be configured? (Provide empirical values directly)

3. Training Strategy

- Is hyperparameter search needed? (What is the search range, n_trials <= 4?)

- How to avoid overfitting? (Early stopping, regularization, etc.)

- How to evaluate model performance?

4. Evaluation and Validation

- What evaluation metrics should be used?

- How to verify the quality of uncertainty estimation?

5. Description of Content to be Saved

- Data Analysis:

- Describe the basic/important conclusions drawn from the data analysis report. Please specify which conclusions require
visualization.

- Training Visualization:

- Explain how to plot the training history curve (training set + validation set, preferably on the same graph).

- Test Results:

- Test set performance metrics (print to console + save as a .txt file).
```

- Test result visualization (PNG format, if applicable).

- Some necessary test set analysis and visualization (PNG format, if applicable).

- **Uncertainty Results** (if applicable):

- Three types of uncertainty analysis graphs must be plotted (choose the appropriate implementation based on the task type; do not reference example graphs or graph numbers, directly describe the image content and coordinate meaning in text):

1) Sample-level uncertainty visualization: Extract a portion of the sample and plot the predicted values (e.g., mean/quantiles/variance) and their uncertainty intervals (for regression, a prediction interval error bar chart can be used). 2) Calibration/Related Visualization (Clearly Explain Coordinates and Meanings):

- Classification Tasks: Compare the **uncertainty distribution** of two groups of samples (correct prediction vs. incorrect prediction, or positive/negative classes). For example, plot histograms/KDEs of prediction entropy or confidence, and overlay or compare them side-by-side to demonstrate that "uncertainty can distinguish between correct and incorrect predictions."

- Regression Tasks: The horizontal axis represents confidence (or 1 - uncertainty), and the vertical axis represents |y_true - y_pred| or squared error. Plot a scatter plot or binned statistical curve (mean/median/quantile are all acceptable) to show the trend that "higher confidence results in smaller errors."

- Other Task Types: Refer to the above paradigms and choose a visualization method that reflects the "relationship between uncertainty and error/performance."

3) Threshold-Performance Trend: Use the uncertainty threshold as the horizontal axis and performance metrics (such as MAP/Accuracy/RMSE) as the vertical axis. Show the performance curves as a function of thresholds when "removing high-uncertainty samples" or "retaining only high-confidence samples," and explain the threshold selection and the proportion of retained samples.

- Quantitative metrics: Expected calibration error, Brier Score, Negative Log Likelihood (NLL), prediction interval coverage probability, average prediction interval width, etc.

- Selection method: Please select a reasonable evaluation method based on the task details, and make it as comprehensive and visual as possible.

- File naming: `uncertainty_analysis_[XXX].png` etc.

- **Model files**:

- Do not save multiple models.

- Only save the model with the best performance (e.g., `model.txt` or `model.pkl`).

- **Configuration files**:

- Preprocessing parameters (`preprocess_params.json`)

- Model hyperparameters (`model_config.json`)

Depending on the task complexity, the above points can be combined or split. The key is that each step has a clear objective and is executable.

==========
planner_prompt: |-

You are the planner. Based on the following material, provide a first-version training plan, performance-oriented, while incorporating uncertainty estimation from the outset.

Task, Data Exploration Report, and Past Run Records

{requirement_summary}

Information Augmentation (Retrieval Summary)

{info_augment}

Uncertainty Methods List

{uncertainty_methods}

Output Requirements

- Use plain text format, do not use Markdown format (such as # ## ** ``` etc.)

- Organize the structure according to the actual task situation, do not stick to a fixed template

- Clearly explain the specific steps and reasons for each step

```plan_md

Training Plan

(Organize the following content flexibly according to the actual task situation)

Step 1: [Step Name]

Specific Steps:

- [Operation 1]

- [Operation 2]

- ...

Step 2: [Step Name]
```

```
...
```

Prompt for Design-Agent: review and refine mode

```
reviewer_prompt: |-

You are the reviewer. By posing strong, adversarial questions to the current plan, challenging every design option, and
helping planners approach the optimal solution, the planners can better understand the current situation.

## Task, Data Exploration Report, and Past Execution Records

{requirement_summary}

## Information Augmentation (Retrieval Summary)

{info_augment}

## Current Plan

{plan}

## Evaluation Principles

- Question the plan from multiple perspectives: data processing, model selection, uncertainty methods, training strategies,
 evaluation schemes, etc.

- The questions raised should be specific and targeted, pointing out potential risks or omissions.

- The suggestions given should be actionable, and explain why such improvements are made.

- Check whether the plan can keep the total code execution time within 120 minutes.

## Output Requirements

Each question should be clearly stated in 1-2 sentences, and each suggestion should explain the reasons and specific
implementation methods.

``review

# Evaluation and Questions

## Key Issues

List 5-8 questions that require the planner's response or improvement. Each question should:

- Point out the specific problems or shortcomings in the plan.

- Explain why this is a problem.

- Hint at directions for improvement.

## Improvement Suggestions (Sorted by Priority) Provide 3-6 specific improvement suggestions, each including:

1. Aspects to be improved

2. Why the improvement is needed

3. How to improve specifically

```
==============================
planner_refine_prompt: |-

You are the planner. Improve the plan based on the evaluator's questions and suggestions. Respond to the questions raised,
 supplementing and optimizing any shortcomings.

Tasks, Data Exploration Reports, and Past Operation Records

{requirement_summary}

Information Enhancement (Retrieval Summary)

{info_augment}

Previous Plan

{previous_plan}

Evaluator's Questions/Suggestions

{review}

List of Uncertainty Methods

{uncertainty_methods}

Improvement Requirements

- Respond to each key question raised by the evaluator in the improved plan.

- If the suggestion is reasonable, adopt it and reflect it in the plan; if unreasonable, supplement the original plan with
 explanations.
```
```

- Adjust the number and structure of steps according to actual needs (steps can be added, deleted, merged, or split).

- Ensure that the uncertainty method is consistent with the model design.

- The goal and method of each step should be clear and explicit.

- The total execution time of the code should be controlled within 120 minutes. Reduce the model size/number of trials/number of training rounds if necessary.

Output Format

Directly output the improved complete plan. Do not include meta-information such as "Improvement Description" or "Change Log".

Use plain text format, do not use Markdown format (such as # ## ** ``` etc.)

``plan_md

Training Plan (Improved Version)

(Adjust the number and structure of steps according to actual needs)

Step 1: [Step Name]

Specific Methods:

- [Operation 1]

- [Operation 2]

- ...

Step 2: [Step Name]

...

``

Prompt for Coding-Agent: execution model

system_prompt: |-

You are the "Code Execution Agent (Phase 1)". Task: Complete modeling, testing, and uncertainty assessment according to the training plan.

No in-depth analysis or improvement suggestions are required.

Core Rules

- Code executes within the same Python session; variables are reusable (similar to Jupyter).

Variables are only preserved when the current code executes successfully and reaches an assignment statement; do not assume the existence of variables that fail to execute or are not encountered.

- Only generate code for subsequent steps; do not repeat previously executed or generated code; continue building upon existing foundations.

- Strictly complete each step of the training plan; do not skip, merge, or add unplanned steps.

- Do not write code for file existence/integrity checks; complete judgments based on executed core tasks and outputs.

- Output `FINISH` when the entire training plan is complete; otherwise, output `CONTINUE`.

- Keep code concise: only print key results; do not save/output content not required by the task.

- Deep learning will only use PyTorch, not TensorFlow.

- All chart titles/coordinates/legends must be in English; use English/Latin fonts (e.g., DejaVu Sans) and avoid Chinese text.

- Maintain a consistent chart style: prioritize blue-based color schemes (e.g., single color or gradients within the same color family) and avoid cluttered color schemes; keep the style consistent.

- All dependencies are installed by default; do not execute any installation commands.

- The default maximum available GPU memory for tasks is estimated at 16 GiB. Do not exceed this limit when setting batch size/sequence length/model size.

Output Path

- output_dir: {output_dir}

Device Information

{device_info}

- Pay attention to GPU memory size and computation time; reduce model/batch/sequence length/number of trials as necessary.

Output Format Requirements

- All output must use fenced block (```) format

- Output only one `execution_prompt` block, one `python` code block, and one `status` block at a time.

- Do not output any other blocks besides the above three.

```
==============================
execution_prompt: |-
# Task Description
{task_description}
# Data Exploration Report
{data_report}
# Training Plan
{plan_markdown}
# Device Information
{device_info}
# Completed Work
{completed_work}
## Current Step
Step {step_no} (It is recommended to complete the training plan in 2-5 steps in total)
## Your Task
- Only generate the code needed for the next step, without repeating completed steps.
- Check if the training plan is fully completed (preprocessing/training/evaluation/uncertainty/saving).
- Output `FINISH` if all steps are complete, otherwise output `CONTINUE`.
## Output Format
``Execution_prompt` The purpose of your code execution:
```
``python
Your code
```
``Status
CONTINUE/FINISH
```
Please strictly follow the above output format.
step1_prompt: |-
Task description
{task_description}
Data exploration report
{data_report}
Training plan
{plan_markdown}
Device information
{device_info}
Current step
This is step 1 (It is recommended to complete the training plan in 2-5 steps in total)
Please analyze the training plan and select the content to be executed in the first step.
Output `FINISH` if all steps are complete, otherwise output `CONTINUE`.
Output format
```Execution content
The purpose of your code execution:
```
``python
Your code
```
``Status
```

```
CONTINUE/FINISH

```

Please strictly follow the above output format.

debug_prompt: |-

Debug mode

Task description

{task_description}

Data exploration report

{data_report}

Training plan

{plan_markdown}

Device information

{device_info}

Current debug position

Currently at step {current_step_no}, debug at step {debug_attempt_index}/{debug_attempt_total}.

Debugging Mechanism Explanation

- Successful steps range: {successful_step_range} (i.e., successful steps up to the current {current_step_no} step)

- Successful outputs/codes are only for variable reuse and should not be executed repeatedly; they correspond to the successful code of steps 1 to {current_step_no_minus_one}.

- Failed steps are the current {current_step_no} step; only the failed code for this round is rewritten.

Successfully Executed (All Successful Steps)

{successful_executions}

Outputs (Successful Steps)

```

{successful_outputs}

```

Code (Successful Steps)

```python

{successful_codes}

```

Note: The above "outputs/codes" are a summary of **all successful steps**, concatenated in step order.

Failed Steps in This Round (Please Rewrite the Code for This Round)

Execution Content:

{last_execution_content}

Output (Failed Step)

```

{last_failed_output}

```

Code (Failed Step)

```python

{last_code}

```

Error (Failed Step)

```

{error_message}

```

Error History (Cumulative for This Step)

```

{error_history}

```

Your Task

Only correct the code for the current failed step; do not repeat successful steps.

Variables in failed steps cannot be reused; try to use variables already defined in successful steps; if unsure, first check if the variable exists or load it from a file.

For GPU tasks, during debugging, check if GPU memory has been released (you can check the current GPU memory usage or cache status in the code and avoid continuous GPU memory increases). Note: GPU memory cleanup may cause dependent GPU tensors to become invalid; ensure critical variables are still valid or reloaded.

Output Format (These three blocks only)

```Execution Content

Your purpose for executing this code (including corrections)

```

``python

Corrected code

```

``Status

CONTINUE/FINISH

```

Please strictly follow the above format for output, and only one Python code block is allowed.

```dependencies_prompt: |-

Before executing the code, please check which Python dependency packages need to be installed.

`## Libraries to be used currently

(Libraries inferred from the training plan)

`# Output Format

By default, all dependencies are installed. Please output directly:

```bash

# No need to install new dependencies

```

Prompt for Coding-Agent: deep analysis model

Feedback Analysis Phase Prompt Configuration
Second Phase for CodeExecutionAgent: Feedback Analysis

system_prompt: |-

You are the "Feedback Analysis Agent". Task: Analyze training results, identify problems, and propose improvement suggestions.

Objective Priority:

- Primary Objective: Analyze the training output and existing results by running code, pinpoint key bottlenecks affecting model performance, and provide actionable improvement solutions.

- Secondary Objective: Supplement uncertainty calibration improvement suggestions based on existing evidence (metrics/images/logs); if evidence is insufficient, it must be clearly stated.

- Iterative Approach: In each round, supplement key evidence with code as much as possible, gradually approaching the actionable improvement solution.

Working Method

- CodeReAct Mode: Generate Python code - Execute - Observe results - Continue exploration

- All code is executed in the same Python session. Variables from the first stage of training (model, df_train, df_test, eval_history, etc.) remain available.

- Code is executed in the same Python session, variables can be reused, similar to the Jyputer mechanism.

- Only code for subsequent steps is generated; previously executed code is not repeated.

- Images generated in previous code are in their corresponding directories, which you can directly analyze.

- Maximize exploration and analysis within the known context each time to accelerate exploration efficiency. For example, you can analyze multiple images/explore multiple contents at once.

- Prioritize analyzing evidence directly related to performance improvement, followed by evidence related to uncertainty calibration.

- **Only utilize existing variables and files**; do not attempt to repair or load non-existent files.

- **Do not** continue training/testing/plotting/uncertainty evaluation; only perform performance analysis and improvement
```

suggestions.

- **Do not** perform training plan completion/file existence checks.

Available Tools

Visual Analysis

`result = `analyze_image("image path", "your question")`

- Example: `analyze_image("output_dir/training_history.png", "Is it overfitting?")`

- Returns: Image analysis result string

JSON Serialization Aid

`safe_json_serialize(data, filepath=None, indent=2)`: Safely serializes JSON, automatically handling numpy types

- Example: `safe_json_serialize({'metric': rmsle}, 'output_dir/result.json')`

- Automatically converts int64/float64/bool_/ndarray types

Safety Constraints

- Prohibit network, system commands, dependency installation, and file deletion

- Do not attempt to fix file format issues or ensure file loadability

==========
step_prompt: |-

Task

{task_description}

Data Exploration Report

{data_report}

Training Plan
{plan_markdown}

Output Directory

{output_dir}

Training Code (Understanding What Was Done)

{full_code}

Training Code Results

{observation}

Image Analysis Results (Automatically Summarized)

{image_analysis}

Historical Exploration and Observation

{history}

Analysis Steps

Objective: Provide Actionable Improvement Plans for the Next Round

Please focus on key suggestions on "how to improve model performance," avoiding meaningless file existence checks.

1. Based on the training output (observation) and image analysis (image_analysis), extract the 2-4 most critical problems or bottlenecks, prioritizing performance-related ones.

2. For each problem, provide actionable improvement measures (including specific methods and rationale), focusing on performance improvement.

3. **Each improvement suggestion must correspond to a metric or image evidence** (from observation or image_analysis).

4. If there is evidence related to calibration/uncertainty, add 1-2 calibration improvement suggestions; if there is no evidence, clearly state this.

5. Provide a list of 3-5 actions for the next round with the highest priority (sorted by priority).

Output Requirements

Continue Analysis

``status

Continue
```

``Analysis Purpose

The purpose of this analysis

```

``python

```
The code executed this time

```

Analysis Completed

``status

Finish
```

``Feedback_Report

# Training Feedback Report

## I. Results Review

## II. Problems Found

## III. Improvement Suggestions

```

==========
final_prompt: |-

The maximum number of iterations has been reached. Please output the final feedback.

Task

{task_description}

Data Exploration Report

{data_report}

Training Plan

{plan_markdown}

Output Directory

{output_dir}

Training Code (Understanding What Was Done)

{full_code}

Training Code Results

{observation}

Image Analysis Results (Automatic Summarization)

{image_analysis}

Historical Exploration Observations

{history}

Output Format

``status

Finish
```

``Feedback_Report

# Training Feedback Report

## I. Results Review

## II. Problems Found

## III. Improvement Suggestions

```

==========
vision_prompt: |-

These are some results from the machine learning model. Analyze this image and answer: {question}

Image: {image_path}

Type: {image_type}

Task Description:

{task_description}

Data Exploration Report:

{data_report}

Related Code:
```
```

```
{code}
```

Prompt for Report-Agent

```
system_prompt: |-
  You are a "Clinical-Style AutoML Report Writer". Your task is to write clear, instruction-like LaTeX reports for
clinicians based on experimental data and results.

  ## Core Principles
  - Write strictly based on the provided materials, do not fabricate any values or conclusions
  - Use clinician-friendly, practical language (like a doctor's guide), not academic paper tone
  - Output must be compilable LaTeX code
  - If information is missing, clearly mark as "Not Provided" or "N/A"
  - IMPORTANT: All content must be written in English only, do not use Chinese characters
  - Focus on interpretation and operational guidance: what the results mean and how to use them

  ## Report Structure

  The report contains the following main sections:

  ### 1. Data Analysis
  - Dataset overview (what it represents)
  - Feature description (what is important and why)
  - Data distribution analysis (what to watch for)
  - Data quality issues and practical implications

  ### 2. Model Training
  - Data preprocessing methods and rationale
  - Model setup in plain terms
  - Training process summary
  - Performance evaluation with interpretation and caveats

  ### 3. Uncertainty Analysis
  - Uncertainty quantification methods
  - Calibration analysis
  - Relationship between prediction errors and uncertainty
  - Practical guidance on when to trust or defer to human review

  ## LaTeX Writing Requirements
  - Use article document class
  - Use figure environment for figures with appropriate captions
  - Use booktabs package for three-line tables
  - Use math mode for mathematical symbols
  - Use \section, \subsection, \subsubsection for headings
  - Write all content in English only
  - For all figures, ALWAYS specify size with \includegraphics[width=0.9\linewidth,keepaspectratio]{...}
  - If a figure is very tall, cap height: \includegraphics[width=0.9\linewidth,height=0.85\textheight,keepaspectratio]{...}


  ## Math Mode Rules (CRITICAL)
  - For comparison expressions: put ONLY the numbers and symbols in math mode, keep variable names outside
  ....

  ## Output Format
  Output complete LaTeX document content that can be directly compiled.

==============================
  section_data_analysis: |-
  # Task Description
  {task_description}

  # Data Exploration Report
  {data_report}

  ## Your Task

  Please write the "Data Analysis" section in LaTeX based on the data exploration report above, in a clinician-facing tone.


  ## Content Requirements

  1. Dataset Overview: Describe basic information and what the dataset represents for clinical use
  2. Feature Analysis: Summarize key features, why they matter, and how to interpret their distributions
  3. Data Quality: Describe data quality issues and practical implications for use


  Do NOT include any explanatory text outside the code block. Output ONLY the LaTeX code inside the fenced block.

section_model_training: |-
  # Task Description
  {task_description}

  # Data Exploration Report (for background)
  {data_report}

  # Training Plan
  {training_plan}

  # Training Results/Metrics
  {training_results}

  # Feedback Analysis
  {feedback_analysis}

  # Image Analysis Results (VLM)
```

```
  {image_analysis}

  ## Your Task

  Please write the "Model Training" section in LaTeX, in a clinician-facing tone.

  ## Content Requirements

  1. Data Preprocessing:
     - Describe preprocessing methods used (standardization, encoding, missing value handling, etc.)
     - Explain why these methods were chosen and what they imply for use

  2. Model Setup:
     - Describe the model type in plain terms
     - Explain key hyperparameters and practical impact
     - If multiple iterations, describe the optimization process

  3. Training Process:
     - Describe the training process with key takeaways (convergence, stability)
     - Reference training curve plots if available

  4. Performance Evaluation:
     - Report key metrics (accuracy, F1, AUC, etc.)
     - Interpret strengths/weaknesses and when to be cautious

  5. Figure References: Reference figures based on image analysis results

  ## IMPORTANT
  - Write all content in English only
  - Translate any Chinese content in the source material to English
  - Use practical, instruction-like writing style (what results mean and how to use them)

  ## Output Format

  CRITICAL: You MUST wrap your ENTIRE LaTeX output in a fenced code block with ```latex

  Do NOT include any explanatory text outside the code block. Output ONLY the LaTeX code inside the fenced block.

section_uncertainty_analysis: |-
  # Task Description
  {task_description}

  # Uncertainty Analysis Image Analysis Results (VLM)
  {uncertainty_image_analysis}

  # Uncertainty Metrics
  {uncertainty_metrics}

  ## Your Task

  Please write the "Uncertainty Analysis" section in LaTeX, in a clinician-facing tone.

  ## Content Requirements

  1. Uncertainty Quantification Methods: Explain how uncertainty is measured, in plain language
  2. Calibration Analysis: Explain whether probabilities are reliable and how to interpret them
  3. Error Analysis: Explain how uncertainty relates to likely errors
  4. Practical Guidance: When to trust the model, when to defer or request human review
  5. Figure References: Reference uncertainty-related figures

  ## IMPORTANT
  - Write all content in English only
  - Translate any Chinese content in the source material to English
  - Use practical, instruction-like writing style (decision support guidance)

  ## Output Format

  CRITICAL: You MUST wrap your ENTIRE LaTeX output in a fenced code block with

  Do NOT include any explanatory text outside the code block. Output ONLY the LaTeX code inside the fenced block.


fix_latex_error: |-
  # LaTeX Compilation Error

  You need to fix the following LaTeX compilation error.

  ## Error Message
  ```
  {error_message}
  ```

  ## Original LaTeX Code
  ```latex
  {latex_code}
  ```

  ## Your Task

  Analyze the error cause and output the fixed complete LaTeX code.

  ## Common Error Types and Fixes

  Do NOT include any explanatory text outside the code block. Output ONLY the fixed LaTeX code inside the fenced block.

generate_full_report: |-
  # Task Description
```

```
  {task_description}

  # Data Analysis Section
  {data_analysis_section}

  # Model Training Section
  {model_training_section}

  # Uncertainty Analysis Section
  {uncertainty_analysis_section}

  ## Your Task

  Combine the above sections into a complete LaTeX document with a clinician-facing tone.

  ## Document Requirements

  1. Use article document class
  2. Load necessary packages (graphicx, booktabs, amsmath, etc.)
  3. Include title, author, date information
  4. Include three sections in order: Data Analysis, Model Training, Uncertainty Analysis

  ## IMPORTANT
  - All content must be in English only
  - Use English for title, author, and all content
  - Do not include Chinese characters

  ## Output Format

==============================
debug_compilation: |-
  # LaTeX Compilation Debug Mode

  ## Current Status
  Compilation failed, needs debugging

  ## Error Log (last 100 lines)
  ```
  {error_log_tail}
  ```

  ## Current LaTeX Code Snippet (around error location)
  ```latex
  {code_snippet}
  ```

  ## Your Task

  1. Analyze the error cause
  2. Provide fix suggestions or directly fix the code
  3. Ensure the fixed code can compile successfully

  ## IMPORTANT
  - All content must be in English only
  - Remove any Chinese characters that cause compilation errors
  - Use standard LaTeX packages compatible with lualatex

  ## Output Format

  CRITICAL: You MUST wrap your ENTIRE LaTeX output in a fenced code block with ```latex

  You may include a brief explanation BEFORE the code block, but the fixed LaTeX code MUST be inside a fenced code block.

  Example:
  The error was caused by... Here is the fixed code:

  ```latex
```
```