TUSOAI: AGENTIC OPTIMIZATION FOR SCIENTIFIC METHODS

Anonymous authors
Paper under double-blind review

ABSTRACT

Scientific discovery is often slowed by the manual development of computational tools needed to analyze complex experimental data. Building such tools is costly and time-consuming because scientists must iteratively review literature, test modeling and scientific assumptions against empirical data, and implement these insights into efficient software. Large language models (LLMs) have demonstrated strong capabilities in synthesizing literature, reasoning with empirical data, and generating domain-specific code, offering new opportunities to accelerate computational method development. Existing LLM-based systems either focus on performing scientific analyses using existing computational methods or on developing computational methods or models for general machine learning without effectively integrating the often unstructured knowledge specific to scientific domains. Here, we introduce TusoAI, an agentic AI system that takes a scientific task description with an evaluation function and autonomously develops and optimizes computational methods for the application. TusoAI integrates domain knowledge into a knowledge tree representation and performs iterative, domain-specific optimization and model diagnosis, improving performance over a pool of candidate solutions. We conducted comprehensive benchmark evaluations demonstrating that TusoAI outperforms state-of-the-art expert methods, MLE agents, and scientific AI agents across diverse tasks, such as single-cell RNA-seq data denoising and satellite-based earth monitoring. Applying TusoAI to two key open problems in genetics improved existing computational methods (40% power improvement to scDRS in associating cells to disease in simulations and 10.5% enrichment improvement to pgBoost for identifying ground-truth variant-gene pairs) and uncovered novel biology, including 9 new associations between autoimmune diseases and T cell subtypes (e.g., primary biliary cirrhosis with central memory T cells) and 7 previously unreported links between disease variants linked to their target genes (e.g., glucose/HbA1c risk variant rs138917529 with *GCK*). Our code will be publicly available upon publication.

1 INTRODUCTION

Scientific discoveries are often bottlenecked by the slow, manual development of computational tools needed to analyze experimental data. For example, genetics studies have uncovered tens of thousands of disease-associated variants, yet robust computational methods are critically needed to harmonize multi-modal, multi-scale data and uncover the underlying mechanisms (Lappalainen & MacArthur, 2021). Developing such tools is slow and costly because scientists must iteratively (i) review extensive literature, (ii) test modeling and scientific assumptions against empirical data, and (iii) implement these insights into efficient, scalable code. For instance, building robust computational methods to link enhancers with target genes from single-cell multiome data has taken multiple expert groups many years (Dorans et al., 2025), hindered by challenges such as *cis*-regulatory modeling, latent confounding, noisy data, and computational scalability. Large language models (LLMs) have demonstrated strong capabilities in performing human-like analysis (Luo et al., 2025), such as synthesizing relevant literature (Asai et al., 2024), reasoning about biological and modeling assumptions using empirical data (Gao et al., 2024), and generating efficient, domain-specific code (Rasheed et al., 2025). Integrating LLMs with scientific domain knowledge and iterative data exper-

imentation holds great promise to accelerate computational method development, thereby advancing discoveries in science and medicine.

Existing work has produced general-purpose AI agents across scientific domains, including biomedicine (Huang et al., 2025; Jin et al., 2025) and chemistry (M. Bran et al., 2024). These systems primarily focus on performing scientific data analyses rather than developing new computational methods; the former involves assembling and executing pipelines of data formatting and existing tools, whereas the latter requires creating new algorithms or models for specific pipeline steps, involving substantial design, optimization, and validation. In parallel, several studies have developed machine learning engineering (MLE) agents that can design new algorithms for general ML applications (Guo et al., 2024; Trirat et al., 2024; Jiang et al., 2025; Nam et al., 2025), but these approaches do not address domain-specific challenges inherent in scientific research. Developing *AI agents for scientific method development* that integrate structured domain knowledge and systematically explore data-specific assumptions has considerable potential to accelerate the creation of robust computational methods for science and medicine.

Here, we introduce TusoAI, an agentic AI system that takes a scientific task description with an evaluation function, and autonomously develops and optimizes computational methods for the application (Figure 1). TusoAI integrates structured domain knowledge with iterative, domain-specific optimization and model diagnosis, improving performance over a pool of candidate solutions. We demonstrate that TusoAI achieves superior performance across a range of algorithmic, statistical, machine learning, and deep learning applications in science. Our key contributions are:

1. We develop TusoAI, an AI agent specifically tailored for scientific method discovery by integrating structured domain knowledge.
2. We propose a novel framework, featuring (i) knowledge tree for structured representation of domain knowledge, (ii) hierarchical planning with Bayesian updates to balance solution quality and diversity, and (iii) fine-grained generation that integrates model optimization with diagnostic feedback.
3. We benchmark TusoAI on 6 single-cell analysis tasks and 5 scientific deep learning tasks, consistently outperforming baseline methods and frequently surpassing existing expert-designed algorithms with on average 16% improvement in single-cell tasks.
4. Applying TusoAI to two key open problems in genetics improved existing computational methods (40% power improvement to scDRS in associating cells to disease in simulations and 10.5% enrichment improvement to pgBoost for identifying ground-truth variant-gene pairs) and uncovered novel biology, including 9 new associations between autoimmune diseases and T cell subtypes (e.g., primary biliary cirrhosis with central memory T cells) and 7 previously unreported links between disease variants linked to their target genes (e.g., glucose/HbA1c risk variant rs138917529 with *GCK*).

1.1 Related work

LLM-based scientific AI agents. Several works have developed general-purpose AI agents capable of autonomously executing various scientific research tasks. Biomni (Huang et al., 2025) provides a unified agentic environment with tools and databases spanning 25 biomedical domains, integrating LLM reasoning with retrieval-augmented planning and code execution to compose complex workflows. Stella (Jin et al., 2025) employs a multi-agent architecture for autonomous biomedical data analysis, achieving self-evolution by dynamically updating its template library and tool collection. ChemCrow (M. Bran et al., 2024) is a chemistry-focused agent that integrates 18 expert-designed tools and follows the "Thought, Action, Action Input, Observation" format to iteratively reason toward answers. These methods emphasize end-to-end data analysis with established tools, whereas our work focuses on developing new computational methods for domain-specific tasks. Other works have leveraged LLMs to develop application-specific methods, such as single-cell perturbation prediction (Tang et al., 2025), diagnosis prediction (Tan et al., 2025), and mathematical discovery (Romera-Paredes et al., 2024). In contrast, TusoAI targets computational method development across scientific tasks. InternAgent (Team et al., 2025) and its precursor Dolphin (Yuan et al., 2025) iteratively evolve and implement research ideas through an optimization process augmented with literature review. As a concurrent effort, Aygün et al. (2025) combine LLMs with tree search and existing model ensembles to improve scientific algorithms, addressing a similar problem but

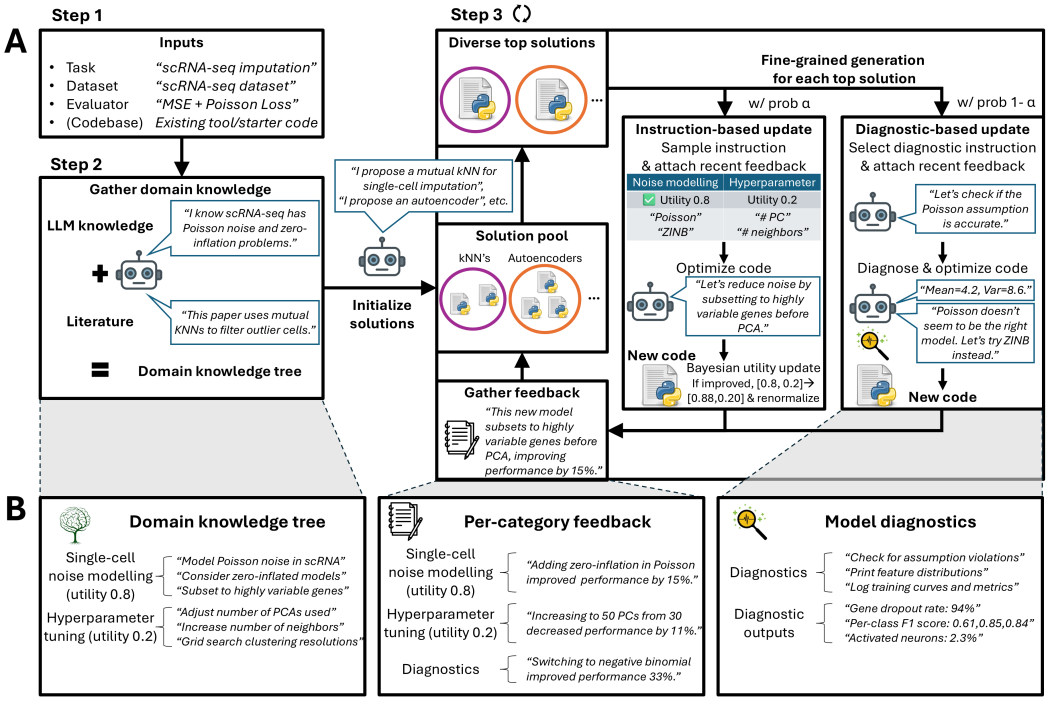

Figure 1: **Scientific method development with TusoAI. (A)** Method overview. **(B)** Example domain knowledge tree (categories and instructions per category), feedback, and diagnostics.

with a different approach from ours, which integrates a domain knowledge tree with fine-grained iterative optimization and Bayesian updates. As their code is not publicly available, direct comparison is not possible.

LLM-based general machine learning agents. Several recent works have developed AI agents for general machine learning engineering. AIDE (Jiang et al., 2025) frames ML engineering as a code optimization problem, combining an LLM with tree search to iteratively improve solutions. R&D Agent (Yang et al., 2025) similarly explores ML architectures in a dynamic feedback loop. DS-Agent (Guo et al., 2024) combines an LLM with case-based reasoning (CBR), retrieving potentially successful solutions from top-ranked Kaggle solutions, and refining them through iterative optimization. MLE-STAR (Nam et al., 2025) retrieves candidate models from the web to form an initial solution, then improve it by targeting specific ML components and ensembling. AutoML-Agent (Trirat et al., 2024) employs retrieval-augmented planning and multi-agent coordination to generate an optimal plan, but executes the plan once without iterative refinement. These methods are less suited to scientific method development, where domain knowledge is unstructured, existing ML models may be unavailable, and search spaces are continually evolving. We address these challenges through structured domain knowledge representation and hierarchical planning with Bayesian updates during iterative optimization.

Classical automatic machine learning (AutoML) frameworks. Classical (non-LLM) AutoML frameworks aim to construct high-performing ML models from scratch by searching over key components such as feature preprocessing, model architectures, hyperparameters, and pipeline composition. Notable examples include auto-sklearn (Feurer et al., 2015), H2O (LeDell et al., 2020), AutoGluon (Erickson et al., 2020), and TPOT (Olson & Moore, 2016). Within deep learning, neural architecture search (NAS) methods specialize in optimizing neural architectures, with examples such as DARTS (Liu et al., 2018) and AMBER (Zhang et al., 2021). While effective for standard ML tasks, these approaches are constrained by predefined search spaces and are less suited to scientific domains, where domain knowledge and optimization objectives are unstructured and continually evolving, making LLM-based agents a more natural fit.

2 PROBLEM FORMULATION

We consider the problem of automatic scientific algorithm optimization with LLMs. Given a general solution space $\mathcal{S}^{\text{full}}$ (e.g., all Python scripts) and an evaluator $h(\cdot) : \mathcal{S}^{\text{full}} \mapsto \mathbb{R}$, the objective is to find the optimal solution $s^* = \arg\max_{s \in \mathcal{S}^{\text{full}}} h(s)$. $h(\cdot)$ can be any evaluation metric, such as AUC, average of several metrics, or domain-specific measures (e.g., enrichment of inferred disease genes against an expert-curated set). We assume access to a task description $\mathcal{T}$ (e.g., "single-cell RNA-seq imputation"), a domain-specific knowledge base (e.g., scientific papers), and a general LLM that can be instantiated as agents. The agent can, for example, summarize domain priors from $\mathcal{T}$, retrieve information from the knowledge base, and refine a candidate solution s based on instructions. The goal is to iteratively implement and improve solutions to maximize $h(\cdot)$ within a time budget. We consider two settings: a *cold start*, where optimization begins from scratch, and a *warm start*, where an initial solution s_{init} (e.g., a state-of-the-art method) is given for further improvement.

3 METHODS

TusoAI takes as input a task description $\mathcal{T}$, a dataset $\mathcal{D}$, an evaluator $h(\cdot)$, and optionally an initial solution s_{init}. It outputs an optimized solution s^* (Algorithm 1). Developing computational methods for scientific domains poses several challenges. First, domain-specific knowledge is often unstructured, which we address using a knowledge tree that organizes information into categories and within-category instructions. Second, approaches and optimization strategies can vary widely, which we manage through hierarchical planning with Bayesian updates to promote diversity while ensuring solution quality. Third, understanding complex data patterns is challenging, which we mitigate with fine-grained generation that integrates model optimization with diagnostic feedback.

TusoAI consists of 3 steps. First, it gathers domain knowledge by summarizing key scientific papers, ensuring that optimization instructions reflect established best practices and recent advances rather than relying solely on LLM priors. Second, it builds a two-level knowledge tree of structured instructions: (1) categories of optimization strategies and (2) specific instructions within each category, promoting both diversity and relevance. Categories and instructions are first drafted by the LLM and then refined through additional LLM queries in conjunction with paper summaries to ensure diversity and scientific rigor; we also predefine a diagnostic category $\mathcal{I}_{\text{diag}}$ to guide data logging and model diagnosis. Third, after initializing candidate solutions, it iteratively selects diverse top performers and improves them through either instruction-based or diagnostic-based optimization. Instruction categories are sampled adaptively via a Bayesian strategy informed by past performance, while feedback comparing new and prior solutions helps discourage repetition. Examples of instructions generated are provided at Appendix B.

Step 1: Gather domain knowledge. TusoAI first retrieves up to 10 key papers from Semantic Scholar (Allen Institute for AI, 2025) relevant to $\mathcal{T}$, ranked by citation count. For each paper, an agent A_{paper} creates a 15-point technical summary from the abstract and iteratively refines it using each paragraph of the paper's Methods section (up to 1,200 words to focus solely on technical content without relying on costly deep research agents parsing the entire document). This produces $\mathcal{P} = \{\mathcal{P}_i\}$, where each $\mathcal{P}_i$ is a refined 15-point summary of paper i's method.

Step 2: Build structured instructions. TusoAI uses a draft-then-refine strategy to construct optimization categories, where an agent A_{cate} first drafts candidate categories from the task description $\mathcal{T}$, then refines them by iterating through each paper summary $\mathcal{P}_i \in \mathcal{P}$, adjusting existing categories or adding new ones as needed. Categories are task-specific and can be general (e.g., "regularization", "model architectures") or domain-specific (e.g., "single-cell noise modeling", "genetic feature interactions"). Each category is assigned a probability π_c representing its utility in the optimization process; π_c is initialized by A_{cate} so that tasks earlier in the pipeline (e.g., "feature preprocessing") receive higher weight than later ones (e.g., "hyperparameter tuning"). Similarly, TusoAI uses a draft-then-refine strategy to initialize instructions for each category, where an agent A_{instr} first drafts 10 candidate instructions $\mathcal{I}_c$ from the task description $\mathcal{T}$. These instruction lists are then refined by incorporating 10 additional instructions for each paper summary $\mathcal{P}_i \in \mathcal{P}$. For feedback, TusoAI initializes an empty list $\mathcal{F}_c \leftarrow \emptyset$ for each category, which is updated with category-specific feedback during optimization. A special predefined diagnostic category $\mathcal{I}_{\text{diag}}$ provides instructions for logging diagnostic information useful for model updates, with its own feedback list $\mathcal{F}_{\text{diag}}$.

Algorithm 1 TusoAI

Input: Task $\mathcal{T}$; dataset $\mathcal{D}$; evaluator $h(\cdot)$; optional initial solution s_{init}.
Hyperparameters: Time budget T_{budget} (default 8 hrs).
Gather domain knowledge: $\mathcal{P} \leftarrow A_{\text{paper}}(\mathcal{T})$ ▷ Paper summaries
Build structured instructions:
  $\mathcal{C}, \{\pi_c\}_{c\in\mathcal{C}} \leftarrow \text{DraftThenRefine}(A_{\text{cate}}, \mathcal{T}, \mathcal{P})$ ▷ Instruction categories with probabilities
  For each $c \in \mathcal{C}$: ▷ Per-category instructions and feedback
    $\mathcal{I}_c \leftarrow \text{DraftThenRefine}(A_{\text{instr}}, \mathcal{T}, \mathcal{P}, c)$, $\mathcal{F}_c \leftarrow \emptyset$
  $\mathcal{I}_{\text{diag}} \leftarrow \emptyset$, $\mathcal{F}_{\text{diag}} \leftarrow \emptyset$ ▷ Diagnostic instructions and feedback
Initialize solutions: $\mathcal{S} \leftarrow A_{\text{init}}(\mathcal{T}, \mathcal{P}, s_{\text{init}})$; $N_{\text{top}} \leftarrow |\mathcal{S}|$
While wall-clock time $< T_{\text{budget}}$ **do**
  Select N_{top} diverse top solutions from $\mathcal{S}$
  for each top s **do**
    if Bernoulli(α) **do** ▷ Instruction-based optimization, defualt $\alpha = 0.8$
      Sample $c \sim \text{Cat}(\{\pi_c\}_{c\in\mathcal{C}})$; optimize $s' \leftarrow A_{\text{optim}}(s, \mathcal{I}_c, \mathcal{F}_c)$
      if $h(s') > h(s)$ **do** $\pi_c \leftarrow 1.1\pi_c$; renormalize $\{\pi_c\}_{c\in\mathcal{C}}$ ▷ Update category utility
      $\mathcal{F}_c \leftarrow \mathcal{F}_c \cup \{A_{\text{feedback}}(s, s')\}$ ▷ Gather category-specific feedback
    else ▷ Diagnostic-based optimization
      $s' \leftarrow A_{\text{diag}}(s, \mathcal{D}, \mathcal{I}_{\text{diag}}, \mathcal{F}_{\text{diag}})$ ▷ Get model&data log info then optimize
      $\mathcal{F}_{\text{diag}} \leftarrow \mathcal{F}_{\text{diag}} \cup \{A_{\text{feedback}}(s, s')\}$ ▷ Gather diagnostic feedback
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s'\}$
  $N_{\text{sol}} \leftarrow \max(1, N_{\text{top}} - 1)$ every 2 rounds
return $s^* \in \arg\max_{s\in\mathcal{S}} h(s)$

Step 3.1: Initialize solutions. The initialization agent A_{init} drafts 5 candidate solution descriptions from $\mathcal{T}$ and iteratively refines them using each paper summary in $\mathcal{P}$, adding new descriptions or improving existing ones (e.g., "zero-inflated Poisson with kNN smoothing"). These are basic descriptions designed to start from scratch and explore to avoid simply re-implementing existing solutions. It then attempts to implement and debug each solution; those that successfully compile form the initial solution pool $\mathcal{S}$. Each implementation attempt is limited to 10 minutes with up to 4 bug-fix attempts.

Step 3.2: Iterative optimization. Given the current solution set $\mathcal{S}$, TusoAI selects diverse top solutions by clustering them based on code-text similarity and, within each cluster, choosing the shortest solution whose performance is within 0.1% of the cluster's best model; this helps discourage overfitting and randomness while maintaining diversity and concise code. For each cluster's top solution s, TusoAI performs either instruction-based optimization (80% probability) or diagnostic-based optimization (20% probability). The resulting solution s' is added to the pool $\mathcal{S} \leftarrow \mathcal{S} \cup s'$. Each implementation attempt is limited to 10 minutes with up to 2 bug-fix attempts. This time regularization ensures the optimization period is not wasted on a few inefficient implementations, and encourages the final method to be scalable.

- **Instruction-based optimization.** The optimization agent A_{optim} selects an instruction by first sampling an instruction category $c \sim \text{Cat}(\{\pi_c\}_{c\in\mathcal{C}})$, then uniformly draw 3 candidate instructions from $\mathcal{I}_c$, and finally choosing the most promising among them. It then optimizes s to produce s' using the selected instruction in conjunction with 5 most recent feedback entries from $\mathcal{F}_c$. If $h(s') > h(s)$, TusoAI updates the category utility by setting $\pi_c \leftarrow 1.1\pi_c$ and renormalizing $\{\pi_c\}_{c\in\mathcal{C}}$. Finally, the feedback agent A_{feedback} summarizes the change from s to s' and appends it to $\mathcal{F}_c$ (e.g., "this optimization constructed a kNN on the top 50 PC's rather than on all genes, improving performance by 15%").
- **Diagnostic-based optimization.** The optimization agent A_{optim} selects an instruction by first uniformly draw 3 candidate instructions from $\mathcal{I}_{\text{diag}}$ and then choosing the most promising among them (e.g., "training curves", "distribution checks", "validation of assumptions"). It then diagnoses and improves s to produce s' using the selected instruction in conjunction with 5 most recent feedback entries from $\mathcal{F}_c$: it runs s to collect diagnostic logs and then uses this information to produce an improved model s'. Finally, the feedback agent A_{feedback} summarizes the change from s to s' and appends it to $\mathcal{F}_{\text{diag}}$.

4 EXPERIMENTS

We evaluate TusoAI on 11 scientific applications spanning diverse domains, including 6 single-cell analysis tasks (Luecken et al., 2025) and 5 scientific deep learning tasks (Tu et al., 2022). The single-cell tasks include denoising (Denoise), cell-type label projection (Label), batch integration (Batch), identification of spatially variable genes (SVG), decomposition of spot-level spatial data into specific cell types (Decomp), and dimensionality reduction for visualization (Visual). The scientific deep learning tasks include omnidirectional vision (Spherical), prosthetics control (NinaPro), medical diagnostics (ECG), earth monitoring (Satellite), and genetic prediction (DeepSea). In each task, we run TusoAI for 8 hours, optimizing performance on a validation dataset, and evaluating final performance on a separate deployment dataset. We conduct comprehensive ablation studies to assess the contribution of different components of TusoAI (Subsection 4.2), and two case studies demonstrating how TusoAI can reveal new biological insights in genetics (Section 5). Additional details are provided in Appendix C, D, J, K.

Baseline methods. We compare TusoAI against the state-of-the-art MLE agent AIDE (Jiang et al., 2025), scientific agents Biomni (Huang et al., 2025) and ChatGPT-Agent (OpenAI, 2025), and top-performing application-specific methods. Biomni (LLM backbone Claude-4-Sonnet) and ChatGPT-agent (LLM backbone GPT-5) are used to iteratively build models on data for single-cell tasks; for deep learning tasks, where Biomni and ChatGPT-agent were unable to operate, we substitute the best of ten models constructed by Claude-4-Sonnet and GPT-5. For application-specific baselines, we use the "top-performing expert" method for single-cell tasks (Luecken et al., 2025), and all baseline methods, including expert models and NAS methods, for the scientific deep learning tasks (Tu et al., 2022). This set of baselines is consistent with related work in scientific optimization (Aygün et al., 2025) and a recent benchmark that identified AIDE and Biomni as top-performers (Miller et al., 2025). For details on baseline implementation, see Appendix E.

4.1 PERFORMANCE ACROSS BENCHMARK EXPERIMENTS

Results for the 6 single-cell tasks and 5 scientific deep learning tasks are reported in Tables 1 and 2. We reached 2 main conclusions. First, TusoAI consistently outperformed baseline methods across benchmarks when generating code from scratch (average rank of 1.2 for single-cell tasks and 2.8 for scientific deep learning tasks, vs. 3.0 and 4.0 for the second best, resp.). Second, the methods constructed by TusoAI are novel rather than simple re-implementations of existing approaches or calls to standard packages. Examples include: (i) in single-cell denoise, TusoAI designed a non-negative matrix factorization (NMF) approach that models dropout rates, Poisson noise, and performs iterative refinement, distinct from the only other NMF-based approach in the OpenProblems benchmark, ALRA (Linderman et al., 2022); (ii) in SVG, TusoAI adapted known techniques such as modeling expression as a function of spatial coordinates and neighborhood summaries to create a custom, high-performing method; (iii) in Satellite, TusoAI combined preprocessing, training procedures, loss functions, and ensembling techniques to build the top-performing model; and (iv) in Spherical, TusoAI fine-tuned layers of ResNet-50 and augmented the data with random flips and rotations. Third, all methods constructed by TusoAI are computationally efficient, owing to the runtime constraints imposed during optimization.

	Denoise	Label	Batch	SVG	Decomp	Visual	Avg	Avg rank
Expert	0.28	0.85	0.71	0.66	0.49	**0.44**	0.57	3.7
AIDE*	<u>0.30</u>	0.87	0.71	<u>0.73</u>	0.06	**0.44**	0.52	<u>3.0</u>
Biomni*	0.16	**0.89**	0.82	0.16	0.53	0.35	0.49	3.7
ChatGPT-Agent*	0.03	0.81	**0.83**	0.60	**0.74**	0.38	<u>0.57</u>	3.5
TusoAI*	**0.35**	**0.89**	**0.83**	**0.80**	<u>0.64</u>	**0.44**	**0.66**	**1.2**

Table 1: **Single-cell benchmarks.** We report performance across 6 single-cell tasks. "*" denotes agentic methods. Best in **bold**, second-best <u>underlined</u>. 95% CIs across 3 random seeds all under 0.01 and thus not shown.

We conducted 2 secondary analyses. First, we assessed the diversity of code produced by TusoAI and AIDE over 8 hours of optimization, quantifying code diversity using cosine similarity of text embeddings between each candidate and its 10 previous and 10 subsequent iterations (Figure 2A).

	Spherical	NinaPro	ECG	Satellite	DeepSEA	Avg	Avg rank
WRN default	0.14 ± 0.01	**0.93** ± 0.00	0.57 ± 0.00	0.85 ± 0.00	0.60 ± 0.00	0.62	6.9
DenseNAS random	0.29 ± 0.02	0.92 ± 0.01	0.58 ± 0.00	0.86 ± 0.00	0.60 ± 0.00	0.65	5.4
DenseNAS original	0.27 ± 0.01	0.90 ± 0.01	0.60 ± 0.00	0.86 ± 0.01	0.60 ± 0.00	0.65	5.8
Perceiver IO	0.17 ± 0.00	0.78 ± 0.02	0.34 ± 0.00	0.84 ± 0.00	0.62 ± 0.00	0.55	9.6
XGBoost	0.03 ± 0.00	0.78 ± 0.01	0.44 ± 0.00	0.64 ± 0.00	0.50 ± 0.00	0.48	11.6
WRN ASHA	0.25 ± 0.00	**0.93** ± 0.01	0.57 ± 0.00	0.84 ± 0.01	0.59 ± 0.00	0.63	7.1
DARTS	**0.52** ± 0.03	0.82 ± 0.01	0.66 ± 0.00	0.87 ± 0.00	0.68 ± 0.00	**0.71**	4.0
AMBER	N/A	N/A	0.67 ± 0.00	0.87 ± 0.00	0.68	N/A	N/A
Expert	0.33 ± 0.01	0.91 ± 0.01	**0.72** ± 0.00	0.80 ± 0.00	0.70 ± 0.00	0.69	4.6
AIDE*	0.16 ± 0.01	0.86 ± 0.00	0.52 ± 0.01	0.83 ± 0.01	0.57 ± 0.00	0.59	9.8
GPT-5*	0.36 ± 0.00	0.89 ± 0.00	0.58 ± 0.03	0.86 ± 0.01	0.66 ± 0.00	0.67	5.8
Claude-4-Sonnet*	0.40 ± 0.00	0.90 ± 0.00	0.50 ± 0.01	0.88 ± 0.00	**0.73** ± 0.00	0.68	4.6
TusoAI*	0.42 ± 0.01	0.90 ± 0.00	0.61 ± 0.00	**0.89** ± 0.01	0.70 ± 0.00	0.70	**2.8**

Table 2: **Scientific deep learning benchmarks.** We report performance across 5 scientific deep learning tasks. "*" denotes agentic methods. Performance of non-agentic methods extracted from NASBENCH-360. Best in **bold**, second-best underlined. 95% CIs provided across 3 random seeds.

TusoAI achieved substantially higher diversity than AIDE throughout the optimization process. For example, in the batch integration benchmark, AIDE repeatedly proposed small variations of UMAP-based dimensionality reduction, whereas TusoAI explored a wide variety of dimensionality reduction, transformation, and scaling techniques. This higher diversity is perhaps due to TusoAI's instruction sampling, feedback, and diagnosis procedures, which encourage diverse solutions. In contrast, AIDE promotes incremental changes at each optimization step to facilitate traceability, which may bias the search toward local tuning rather than full exploration. Details of the diversity procedure is provided in Appendix G. Second, we characterized the optimization trajectory of TusoAI on the single-cell denoising task (Figure 2B). We identified 5 key developments that led to strong performance: (1) introducing NMF, (2) modeling dropout, (3) modeling Poisson noise, (4) adding iterative refinement, and (5) incorporating a sparsity-balancing step. Notably, during optimization, TusoAI generated many methods that reduced performance before converging on high-performing solutions. Together with the feedback mechanism, this broad exploration allowed TusoAI to efficiently search the solution space and identify top-performing methods.

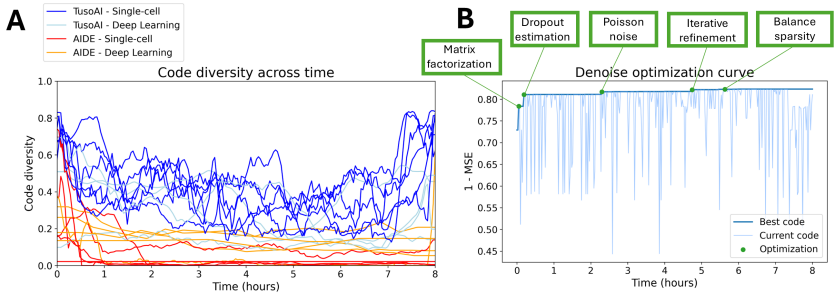

Figure 2: **Behavior of code generated by TusoAI. (A)** Code diversity of TusoAI and AIDE over optimization time, as measured by $1-$ cosine similarity. Each line corresponds to a dataset. **(B)** Performance of the current code and the best code over optimization time for a representative task "Denoise". Key optimization changes with their occurrence times are annotated.

4.2 Ablation Studies

We conducted extensive ablation studies to evaluate the impact of each novel component of TusoAI, including: (i) removing the categorical structure and placing all instructions and feedback into a single category (No categories); (ii) disabling the Bayesian sampling strategy across categories (No Bayesian); (iii) disabling the model diagnosis capability (No diagnosis); and (iv) discarding domain knowledge altogether, such that each iteration simply applies a generic instruction (e.g., "Optimize this model"; No knowledge). Removing these components each negatively affected over-

all performance (Table 3). We attribute this to reduced code diversity (mean diversity 0.48 vs. 0.44/0.39/0.38/0.33 for ablated versions, resp.) and computational efficiency (mean time to optimize 2.3 hours vs. 2.4/3.0/2.6/2.4 for ablated versions, resp.). Removing domain knowledge had the strongest impact on performance and diversity, while removing Bayesian updates most reduced TusoAI's computational efficiency. See Appendix H for further ablation details.

We next assessed the impact of LLM backbones used by TusoAI, testing across 5 different LLMs: low-latency models GPT-4o-mini (default) and Claude-3.5-Haiku; state-of-the-art reasoning models GPT-5 and Claude-4-Sonnet; and open-source GPT-oss-120b. Results are shown in Table **??**. Apart from GPT-oss-120b, TusoAI achieved relatively consistent performance across all LLMs for most tasks, demonstrating robustness. Interestingly, LLMs such as GPT-5 and Claude-4-Sonnet did not consistently outperform their lower-latency counterparts, GPT-4o-mini and Claude-3.5-Haiku. This may be because, while reasoning models can construct highly complex code, their tendency to overbuild (e.g., each of GPT-5's methods are 300+ lines of code) makes subsequent iterations difficult to refine; in contrast, low-latency but capable models like GPT-4o-mini and Claude-3.5-Haiku, when paired with an appropriate system design, performed just as well at a fraction of the cost (e.g., optimizing denoising for 8 hours costs 0.24$ with GPT-4o-mini and 22.3$ with GPT-5). See Appendix I for further LLM details.

	Denoise	SVG	Decomposition	ECG	Satellite	Avg	Avg rank
TusoAI (default)	<u>0.35</u>	**0.80**	<u>0.64</u>	0.61	**0.89**	**0.66**	**2.0**
No categories	0.09	0.72	0.56	<u>0.63</u>	<u>0.86</u>	0.57	3.2
No Bayesian	**0.36**	<u>0.77</u>	0.22	0.57	0.84	0.55	3.4
No diagnosis	0.26	<u>0.77</u>	**0.68**	<u>0.63</u>	<u>0.86</u>	<u>0.64</u>	**2.0**
No knowledge	0.17	0.51	0.07	**0.68**	0.85	0.46	3.8
GPT-4o-mini (default)	0.35	**0.80**	0.64	0.61	**0.89**	0.66	<u>2.2</u>
GPT-5	0.31	**0.80**	**0.82**	**0.67**	0.87	**0.69**	<u>2.2</u>
Claude 3.5 Haiku	**0.41**	0.78	<u>0.70</u>	<u>0.63</u>	**0.89**	<u>0.68</u>	**1.8**
Claude 4 Sonnet	0.32	0.78	0.53	0.59	0.84	0.61	4.2
GPT-oss-120b	<u>0.39</u>	0.74	0.13	0.61	0.85	0.54	3.8

Table 3: **Ablation studies (top) and varying LLM backbone (bottom).** Best in **bold**, second-best <u>underlined</u>.

5 Case Studies in Genetics

We applied TusoAI to address 2 key challenges in genetics: detecting disease-critical cell populations and linking genetic variants to their target genes; these are central to understanding disease etiology but limited by current computational models. We initialized TusoAI with state-of-the-art methods (scDRS (Zhang et al., 2022) and pgBoost (Dorans et al., 2025), resp.) and evaluated its ability to improve these approaches and generate new biological insights.

Detecting disease-critical cell populations. scDRS (Zhang et al., 2022) is a state-of-the-art method that integrates genome-wide association studies (GWAS) with single-cell RNA-seq (scRNA-seq) to identify disease-associated cell populations, but its power is limited by the high noise of scRNA-seq data. Here, we apply TusoAI in conjunction with scDRS and task it with optimizing scDRS's association scoring function. Results are reported in Figure 3. We reached 3 main conclusions. First, the TusoAI-optimized version substantially outperformed the original scDRS in both simulations and real-data benchmarks: it achieved over 40% higher power in causal simulations (Figure 3A) while retaining calibration in null settings (Appendix J), and identified 21% more true cell type–disease associations (17 vs. 14) without false associations in a real-data benchmark (Li et al., 2025). Second, the TusoAI-optimized scoring function is concise and interpretable. It computes association scores in *log–log* rather than *log* space, likely because this transformation better captures polygenic disease signals across many genes, avoiding domination by a few highly expressed genes. This improvement reflects TusoAI's ability to efficiently explore function space: it tested 167 unique versions in 24 hours and at a cost of $0.37, whereas the original authors evaluated fewer than 10 versions over 3 months. Third, applying the TusoAI-optimized scDRS to a T cell

dataset (Cano-Gamez et al., 2020) revealed 26 disease-associated T cell subpopulations (vs. 17 by the original method), including regulatory T cells, central memory T cells, and effector memory T cells associated with primary biliary cirrhosis, consistent with the roles of these T cell populations in autoimmunity (Dominguez-Villar & Hafler, 2018; Seo et al., 2025).

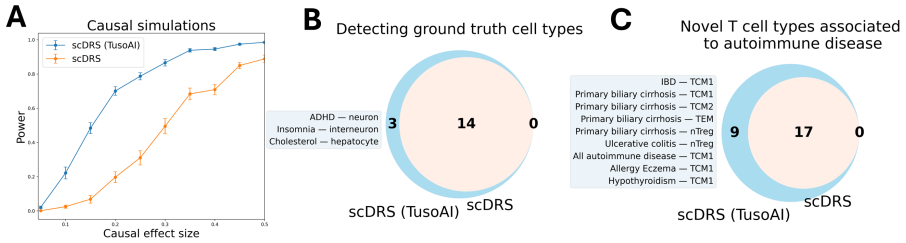

Figure 3: **Optimizing scDRS for detecting cell-disease associations.** **(A)** Assessing power in causal simulations. 95% CI's are calculated across 30 replicates at each perturbation effect size. **(B)** Number of discovered ground-truth trait-cell type pairs at FDR 0.05. **(C)** Number of discovered trait-T cell subtype pairs at FDR 0.05.

Linking genetic variants to genes using single-cell multiome. pgBoost (Dorans et al., 2025) is a state-of-the-art method for linking genetic variants to target genes using single-cell multiome data; it integrates variant–gene distance with multiple linking strategies, but the task remains challenging due to the complexity of genetic regulation (Gazal et al., 2022). Here, we apply TusoAI in conjunction with pgBoost, providing additional positional information for variants and genes, and task it with optimizing distance-based features. Results are reported in Figure 4. We reached 3 main conclusions. First, the TusoAI-optimized model significantly outperformed the original pgBoost, achieving 13.8% higher enrichment of gold-standard links from fine-mapped eQTLs and 7.2% from activity-by-contact (ABC) links, with particularly large gains across longer variant–gene distances (Figure 4A,B). Second, the distance-based features generated by TusoAI are concise and interpretable: 3 are transformed versions of existing features (inverse, squared, and normalized terms), 2 are interactions of gene annotations with distance terms, and the sixth indicates whether the SNP is $<$50kb from the gene's transcription start site, consistent with literature suggesting the typical enhancer–promoter range of around 70kb (Bower et al., 2025). TusoAI discovered these features by testing 511 combinations of 153 novel distance features within 24 hours at a cost of $0.41, whereas the original authors evaluated 5 features over 1.5 months. Third, applying the TusoAI-optimized pgBoost to fine-mapped SNPs for 94 diseases/traits identified 7 novel variant–gene links missed by previous methods. For example, a fine-mapped variant rs138917529 for glucose and HbA1c was linked to *GCK*, consistent with the roles of Glucokinase in regulating glucose levels related to both glucose metabolism (Froguel et al., 1993) and HbA1c variation (Chakera et al., 2015).

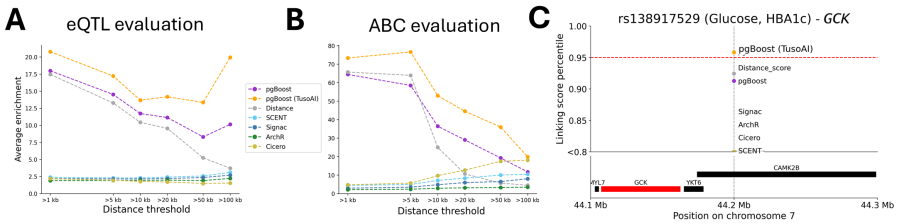

Figure 4: **Optimizing pgBoost for SNP-gene link discovery.** **(A)** Area under the enrichment-recall curve (AUERC, as defined in pgBoost) across distance thresholds for ground truth eQTL variant-gene links. **(B)** AUERC across distance thresholds for ground truth ABC variant-gene links. **(C)** Locus plot of rs138917529 and surrounding genes. Red dashed line indicates cutoff for SNP-gene linking.

REFERENCES

Allen Institute for AI. Semantic scholar. https://www.semanticscholar.org, 2025. Accessed: 2025-09-16.

Akari Asai, Jacqueline He, Rulin Shao, Weijia Shi, Amanpreet Singh, Joseph Chee Chang, Kyle Lo, Luca Soldaini, Sergey Feldman, Mike D'arcy, et al. Openscholar: Synthesizing scientific literature with retrieval-augmented lms. *arXiv preprint arXiv:2411.14199*, 2024.

Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston Anton Kast, Cory Y McLean, Peter Norgaard, Zahra Shamsi, et al. An ai system to help scientists write expert-level empirical software. *arXiv preprint arXiv:2509.06503*, 2025.

G. Bower, E. W. Hollingsworth, S. H. Jacinto, et al. Range extender mediates long-distance enhancer activity. *Nature*, 643:830–838, 2025. doi: 10.1038/s41586-025-09221-6.

Enrique Cano-Gamez, Blagoje Soskic, Theodoros I. Roumeliotis, et al. Single-cell transcriptomics identifies an effectorness gradient shaping the response of cd4+ t cells to cytokines. *Nature Communications*, 11(1):1801, 2020. doi: 10.1038/s41467-020-15543-y. URL https://doi.org/10.1038/s41467-020-15543-y.

Ali J. Chakera, Anna M. Steele, Anna L. Gloyn, Maggie H. Shepherd, Beverley Shields, Sian Ellard, and Andrew T. Hattersley. Recognition and management of individuals with hyperglycemia because of a heterozygous glucokinase mutation. *Diabetes Care*, 38(7):1383–1392, 06 2015. ISSN 0149-5992. doi: 10.2337/dc14-2769. URL https://doi.org/10.2337/dc14-2769.

Margarita Dominguez-Villar and David A. Hafler. Regulatory t cells in autoimmune disease. *Nature Immunology*, 19:665–673, 2018. doi: 10.1038/s41590-018-0120-4. URL https://doi.org/10.1038/s41590-018-0120-4.

Elizabeth Dorans, Karthik Jagadeesh, Kushal Dey, and Alkes L Price. Linking regulatory variants to target genes by integrating single-cell multiome methods and genomic distance. *Nature Genetics*, pp. 1–10, 2025.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages. *CoRR*, abs/2002.08155, 2020. URL https://arxiv.org/abs/2002.08155.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28, 2015.

Philippe Froguel, Habib Zouali, Nathalie Vionnet, Gilberto Velho, Martine Vaxillaire, Fang Sun, Suzanne Lesage, Markus Stoffel, Jun Takeda, Philippe Passa, M. Alan Permutt, Jacques S. Beckmann, Graeme I. Bell, and Daniel Cohen. Familial hyperglycemia due to mutations in glucokinase – definition of a subtype of diabetes mellitus. *New England Journal of Medicine*, 328(10):697–702, 1993. doi: 10.1056/NEJM199303113281005. URL https://www.nejm.org/doi/full/10.1056/NEJM199303113281005.

Shanghua Gao, Ada Fang, Yepeng Huang, Valentina Giunchiglia, Ayush Noori, Jonathan Richard Schwarz, Yasha Ektefaie, Jovana Kondic, and Marinka Zitnik. Empowering biomedical discovery with ai agents. *Cell*, 187(22):6125–6151, 2024.

Steven Gazal, Omer Weissbrod, Farhad Hormozdiari, Kushal K. Dey, Joseph Nasser, Kumar A. Jagadeesh, Daniel J. Weiner, Huwenbo Shi, Charles P. Fulco, Luke J. O'Connor, Bogdan Pasaniuc, Jesse M. Engreitz, and Alkes L. Price. Combining snp-to-gene linking strategies to identify disease genes and assess disease omnigenicity. *Nature Genetics*, 54(6):827–836, June 2022. doi: 10.1038/s41588-022-01087-y. URL https://doi.org/10.1038/s41588-022-01087-y.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Automated data science by empowering large language models with case-based reasoning. *arXiv preprint arXiv:2402.17453*, 2024.

Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, et al. Biomni: A general-purpose biomedical ai agent. *biorxiv*, pp. 2025–05, 2025.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Ruofan Jin, Zaixi Zhang, Mengdi Wang, and Le Cong. Stella: Self-evolving llm agent for biomedical research. *arXiv preprint arXiv:2507.02004*, 2025.

Tuuli Lappalainen and Daniel G MacArthur. From variant to function in human disease genetics. *Science*, 373(6562):1464–1468, 2021.

Erin LeDell, Sebastien Poirier, et al. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020, pp. 24, 2020.

Ang Li, Tian Lin, Alicia Walker, Xiao Tan, Ruolan Zhao, Shuyang Yao, Patrick F. Sullivan, Jens Hjerling-Leffler, Naomi R. Wray, and Jian Zeng. Benchmarking methods integrating gwas and single-cell transcriptomic data for mapping trait-cell type associations. *medRxiv*, 2025. doi: 10.1101/2025.05.24.25328275. URL https://www.medrxiv.org/content/early/2025/06/05/2025.05.24.25328275.

George C. Linderman, Jiajun Zhao, Maria Roulis, et al. Zero-preserving imputation of single-cell rna-seq data. *Nature Communications*, 13:192, 2022. doi: 10.1038/s41467-021-27729-z. URL https://doi.org/10.1038/s41467-021-27729-z.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

M. D. Luecken, S. Gigante, D. B. Burkhardt, et al. Defining and benchmarking open problems in single-cell analysis. *Nature Biotechnology*, 43:1035–1040, 2025. doi: 10.1038/s41587-025-02694-w. URL https://doi.org/10.1038/s41587-025-02694-w.

Xiaoliang Luo, Akilles Rechardt, Guangzhi Sun, Kevin K Nejad, Felipe Yáñez, Bati Yilmaz, Kangjoo Lee, Alexandra O Cohen, Valentina Borghesani, Anton Pashkov, et al. Large language models surpass human experts in predicting neuroscience results. *Nature human behaviour*, 9(2): 305–315, 2025.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, 2024.

Henry E. Miller, Matthew Greenig, Benjamin Tenmann, and Bo Wang. Bioml-bench: Evaluation of ai agents for end-to-end biomedical ml. *bioRxiv*, 2025. doi: 10.1101/2025.09.01.673319. URL https://www.biorxiv.org/content/early/2025/09/07/2025.09.01.673319.

J. M. Mudge, S. Carbonell-Sala, M. Diekhans, et al. Gencode 2025: reference gene annotation for human and mouse. *Nucleic Acids Research*, 53(D1):D966–D975, 2025. doi: 10.1093/nar/gkae1078.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö Arık, and Tomas Pfister. Mle-star: Machine learning engineering agent via search and targeted refinement. *arXiv preprint arXiv:2506.15692*, 2025.

Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pp. 66–74. PMLR, 2016.

OpenAI. Introducing chatgpt agent: Bridging research and action. https://openai.com/index/introducing-chatgpt-agent/, July 2025.

Zeeshan Rasheed, Muhammad Waseem, Kai Kristian Kemell, Aakash Ahmad, Malik Abdul Sami, Jussi Rasku, Kari Systä, and Pekka Abrahamsson. Large language models for code generation: The practitioners perspective. *arXiv preprint arXiv:2501.16998*, 2025.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Eui S. Seo, Sang K. Lee, and Young M. Son. Multifaceted functions of tissue-resident memory t cells in tumorigenesis and cancer immunotherapy. *Cancer Immunology, Immunotherapy*, 74(6): 184, April 2025. doi: 10.1007/s00262-025-04035-x. URL https://doi.org/10.1007/s00262-025-04035-x.

Yanchao Tan, Hang Lv, Yunfei Zhan, Guofang Ma, Bo Xiong, and Carl Yang. Boxlm: Unifying structures and semantics of medical concepts for diagnosis prediction in healthcare. In *Forty-second International Conference on Machine Learning*, 2025.

Xiangru Tang, Zhuoyun Yu, Jiapeng Chen, Yan Cui, Daniel Shao, Weixu Wang, Fang Wu, Yuchen Zhuang, Wenqi Shi, Zhi Huang, et al. Cellforge: Agentic design of virtual cell models. *arXiv preprint arXiv:2508.02276*, 2025.

InternAgent Team, Bo Zhang, Shiyang Feng, Xiangchao Yan, Jiakang Yuan, Runmin Ma, Yusong Hu, Zhiyin Yu, Xiaohan He, Songtao Huang, Shaowei Hou, Zheng Nie, Zhilong Wang, Jinyao Liu, Tianshuo Peng, Peng Ye, Dongzhan Zhou, Shufei Zhang, Xiaosong Wang, Yilan Zhang, Meng Li, Zhongying Tu, Xiangyu Yue, Wangli Ouyang, Bowen Zhou, and Lei Bai. Internagent: When agent becomes the scientist – building closed-loop system from hypothesis to verification, 2025. URL https://arxiv.org/abs/2505.16938.

Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. Automl-agent: A multi-agent llm framework for full-pipeline automl. *arXiv preprint arXiv:2410.02958*, 2024.

Renbo Tu, Nicholas Roberts, Mikhail Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. NAS-bench-360: Benchmarking neural architecture search on diverse tasks. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=xUXTbq6gWsB.

Xu Yang, Xiao Yang, Shikai Fang, Bowen Xian, Yuante Li, Jian Wang, Minrui Xu, Haoran Pan, Xinpeng Hong, Weiqing Liu, Yelong Shen, Weizhu Chen, and Jiang Bian. R&d-agent: Automating data-driven ai solution building through llm-powered automated research, development, and evolution, 2025. URL https://arxiv.org/abs/2505.14738.

Jiakang Yuan, Xiangchao Yan, Shiyang Feng, Bo Zhang, Tao Chen, Botian Shi, Wanli Ouyang, Yu Qiao, Lei Bai, and Bowen Zhou. Dolphin: Moving towards closed-loop auto-research through thinking, practice, and feedback, 2025. URL https://arxiv.org/abs/2501.03916.

Martin Jinye Zhang, Kangcheng Hou, Kushal K Dey, Saori Sakaue, Karthik A Jagadeesh, Kathryn Weinand, Aris Taychameekiatchai, Poorvi Rao, Angela Oliveira Pisco, James Zou, et al. Polygenic enrichment distinguishes disease associations of individual cells in single-cell rna-seq data. *Nature genetics*, 54(10):1572–1580, 2022.

Zijun Zhang, Christopher Y Park, Chandra L Theesfeld, and Olga G Troyanskaya. An automated framework for efficiently designing deep convolutional neural networks in genomics. *Nature Machine Intelligence*, 3(5):392–400, 2021.

A EXAMPLE TUSOAI CODE TEMPLATE

The below code template is for single-cell denoising. The only function TusoAI ever sees and optimizes is the tuso_model function.

Listing 1: Single-cell denoising template file.

```
import scanpy as sc
import pandas as pd
import numpy as np
import scipy as sp
import magic
from anndata import read_h5ad
import scprep
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
from scipy.sparse import issparse
from sklearn.decomposition import PCA
from anndata import AnnData
import random

def mse(adata):
    import anndata
    import scanpy as sc
    import scprep
    import sklearn.metrics

    test_data = anndata.AnnData(X=adata.obsm["test"], obs=adata.obs, var=
        adata.var)
    denoised_data = anndata.AnnData(
        X=adata.obsm["denoised"], obs=adata.obs, var=adata.var
    )

    # scaling and transformation
    target_sum = 10000

    sc.pp.normalize_total(test_data, target_sum=target_sum)
    sc.pp.log1p(test_data)

    sc.pp.normalize_total(denoised_data, target_sum=target_sum)
    sc.pp.log1p(denoised_data)

    error = sklearn.metrics.mean_squared_error(
        scprep.utils.toarray(test_data.X), denoised_data.X
    )
    return error
def tuso_model(adata):

    adata.obsm["denoised"] = ...
    return adata
def main():
    np.random.seed(42)
    random.seed(42)
    adata = read_h5ad('openproblems_datasets/1k_pbmc_processed.h5ad')
    print("tuso_model_start")
    adata = tuso_model(adata)
    print("tuso_model_end")

    val_metric = 1-mse(adata)
    print(f"tuso_evaluate: {val_metric}")

main()
```

B EXAMPLE INSTRUCTIONS GENERATED BY TUSOAI

Listing 2: Example categories for single-cell denoising.

```
['data_preprocessing', 'feature_engineering', 'model_architecture', '
    hyperparameter_tuning', 'imputation_strategies', '
    normalization_methods', 'evaluation_metrics', 'cross_validation', '
    domain_knowledge_integration', 'robustness_techniques', '
    noise_modeling', 'dropout_probability_estimation', '
    graph_neural_network_optimization', 'dropout_pattern_analysis', '
    pipeline_interaction_analysis', 'low_rank_approximation_optimization
    ', 'autoencoder_classifier_integration']
```

Listing 3: Example instructions within a category.

```
 <p>by leveraging graph attention mechanisms to focus on informative cell
      interactions</p>
<p>by incorporating multi-layer graph convolutions to capture
    hierarchical gene expression patterns</p>
<p>by implementing edge dropout to enhance model robustness against noise
     in cell relationships</p>
<p>by utilizing message passing to propagate information across similar
    cell types effectively</p>
<p>by integrating adaptive learning rates for different graph nodes based
     on local connectivity</p>
<p>by employing graph pooling techniques to summarize cellular features
    without losing critical information</p>
<p>by applying graph regularization to maintain structural integrity of
    the cellular network</p>
<p>by utilizing node embeddings to capture latent features of gene
    expression profiles</p>
<p>by optimizing neighborhood sizes dynamically based on data density in
    the graph</p>
<p>by exploring higher-order graph structures to uncover complex
    relationships in RNA-seq data</p>
```

B.1 PREDEFINED DIAGNOSTIC INSTRUCTIONS

Listing 4: Example predefined diagnostic instructions.

```
[
  "by altering or adding diagnostic information to be printed",
  "by altering or adding complex diagnostic information of specific model
       components",
  "by printing key statistical assumptions underlying the model (e.g.,
      independence, normality)",
  "by emitting warnings when model assumptions appear to be violated by
      the data",
  "by logging all implicit assumptions made during model selection or
      preprocessing",
  "by printing assumptions related to feature distributions or
      transformations",
  "by displaying model-specific assumptions such as linearity,
      homoscedasticity, or no multicollinearity",
  "by printing assumptions about data completeness, such as missing value
       tolerance",
  "by logging expectations about input feature scaling or normalization",
  "by displaying prior distributions or regularization beliefs embedded
      in the model",
  "by printing assumptions about label distribution (e.g., class balance
      or stratification)",
  "by emitting diagnostics when data fails to meet i.i.d. (independent
      and identically distributed) assumptions",
  "by logging assumed causal directions or conditional independencies in
      the model",
  "by printing constraints assumed on feature ranges or valid input
      domains",
  "by warning if assumptions about sufficient training data volume are
      not met",
  "by displaying structural assumptions, such as sparsity or low-rank
      representations",
  "by logging assumptions related to stationarity or autocorrelation in
      time-dependent data",
]
```

C SINGLE-CELL ANALYSIS TASKS SETUP

The OpenProblems benchmark (Luecken et al., 2025) contains 12 single-cell analysis tasks with numerous testing datasets and benchmark metrics for each. We select 6 tasks: single-cell denoising, label projection, batch integration, spatially variable gene identification, spatial decomposition of cell types, and visualization. These were selected with the following criteria. First, we required more than one dataset, such that we can optimize on one dataset, and deploy the learned method on the remaining testing datasets, excluding the 2 cell-cell communication tasks and perturbation prediction. Second, a publicly available Github to ensure we are reproducing the testing procedures correctly, excluding multimodal integration and modality prediction. Third, a method for the task should be able to run in a reasonable amount of time on a CPU, excluding the foundation model benchmark.

In each task, we performed optimization on one dataset which could be run in a reasonable amount of time (< 2 for a simple baseline model). The learned methods of each baseline were then applied to the deployment datasets. In selecting benchmark metrics for each task, we had three criteria. First, the metric should not have unavoidable trivial solutions, excluding the Poisson loss metric from denoising, as this can be easily minimized by simply down-weighting lowly expressed genes, including by just scaling genes by their variance or re-normalizing the data. Second, the metrics should be computationally efficient to run, so optimization speed of each method will not be dominated by running metrics. This excluded several metrics from batch integration and visualization. Third, the metric should line up with the task. In the SVG task, it is initially measured in correlation with spatial variability scores, however, the simulation procedure generates binary 0/1 labels of spatial variability, thus we use accuracy of classifying a gene as SVG instead. We also normalize metrics such that each is between 0 and 1 and a higher score is better. The score in denoising is 1-MSE, normalized so that no denoising is 0, and perfect denoising is 1. The score in spatial decomposition is normalized so that a random cell type assignment is 0, and perfect decomposition is 1. See Table 4 for a full breakdown of datasets and metrics used in single-cell tasks.

	Optimization dataset	**Testing datasets**	**Benchmark metrics**
Denoise	1K PBMC	5K PBMC Pancreatic	MSE
Label	5k cells from Immune Cell Atlas	Diabetic Kidney GTEX v9 HypoMap Mouse Pancreatic Islet Atlas Tabula Sapiens	Accuracy F1 macro F1 micro F1 weighted
Batch	5k cells from Immune Cell Atlas	Diabetic Kidney GTEX v9 HypoMap Mouse Pancreatic Islet Atlas Tabula Sapiens	Graph connectivity ASW label ASW batch
SVG	Drosophila Stereo-seq E5	Drosophila Stereo-seq E10 Drosophila Stereo-seq E9 Drosophila Stereo-seq E6	Accuracy
Decomp	TMS Lung (alpha=1.0)	TMS Lung (alpha=0.5) TMS Lung (alpha=5.0) Pancreas (alpha=0.5) Pancreas (alpha=1.0) Pancreas (alpha=5.0)	R^2
Visual	Mouse HSPCT	5K PBMC Mouse Myeloid Zebrafish	Trustworthiness Distance correlation Density Preservation

Table 4: **Single-cell benchmark setup.** Datasets and metrics refer to the setup on the OpenProblems webpage.

D DEEP LEARNING TASKS SETUP

The NASBENCH-360 benchmark Tu et al. (2022) contains 10 deep learning tasks across scientific domains with predefined training, validation, and testing splits, as well as evaluation procedures. We select 5 tasks: Spherical, NinaPro, DeepSEA, Satellite, and ECG. These were selected with the following criteria. First, the task should be scientific and somewhat understudied compared to standard ML tasks, excluding the 2 standard image and audio classification tasks. Second, to ensure fair comparison against the precomputed baselines, we removed tasks where we were uncertain about reproducing the evaluation procedure, partly due to recent GitHub or package updates requiring debugging, excluding Cosmic, PSICOV, and DarcyFlow.

In each task, we performed optimization by training a model on the predefined training set and attaining a score on the validation set. The final testing accuracy of optimized models is attained when deploying the model on the predefined test set. We use the same splits and metrics defined in the original paper.

E BASELINE IMPLEMENTATIONS

AIDE. AIDE takes as input a data folder, task description, and evaluation metric. While originally designed for whole-workflow construction in ML tasks, this can be adapted to general optimization in the following ways. First, AIDE can operate on any data input in the data folder. If specific preprocessing information was needed, we could input this code to the task description. Second, in place of specifying an accuracy metric (e.g., "F1 score"), we instead simply input the entire evaluation function in Python, and found this worked well. As our goal is optimization and not construction, AIDE's initial prompt is tuned until code was consistently generated and optimized upon, typically requiring the same formatting information as other methods. AIDE is run for the same length as TusoAI (8 hours) in the same conda environment on the same CPU (Optimization for AIDE and TusoAI is performed on the same Intel(R) Xeon(R) Gold 5416S.) or GPU (Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz), given 4 threads and 50GB of memory. While AIDE does have a default timeout per execution of 1 hour, on attempting to set this to the same time as TusoAI led to consistent crashes on more of half of tasks, thus we left it as is.

Biomni. We access Biomni through its web page. Biomni runs on a CPU and can take input files up to a limit, has a runtime execution of 1 hour. While not specifically designed for optimization, we can upload the same template code and data as TusoAI then ask Biomni to perform an iterative process of updates. In practice, this led to between 2 and 10 iterations per task between 20 minutes and 4 hours.

ChatGPT-Agent. We access ChatGPT-Agent through its web page. ChatGPT-Agent can accept input files up to 25MB and has a runtime execution of 1 hour. We upload the same template code and data as TusoAI and ask ChatGPT-Agent to perform an iterative process of updates. In practice, this led to between 2 and 7 iterations per task between 10 minutes and 5 hours.

Expert. The expert baselines for NASBench-360 are pre-computed from their paper. For single-cell tasks, we selected the expert method with the following criteria. First, it should be within the top 3 methods as defined by the existing OpenProblems benchmarking. Second, the OpenProblems Github should have code for reproducing this method. Third, we selected the method that was particularly efficient compared to others, if applicable, defined by a runtime of less than 10 minutes on OpenProblems, with others having greater than 1 hour. This left us with the following expert methods, whose code we extracted from the OpenProblems Github:

1. Denoise - MAGIC
2. Batch - Combat
3. Label - Logistic Regression
4. Decomposition - NNLS
5. SVG - SPARK-X
6. Visualize - T-SNE (log10CP10K)

Claude-4-Sonnet and GPT-5. In deep learning tasks where Biomni and ChatGPT-Agent cannot apply due to computational limitations (file size, runtime, GPU access), we substitute the best of 10 models generated by Claude-4-Sonnet and GPT-5. 10 models are generated by prompting these LLMs using the same template that would have been used in Biomni and ChatGPT-Agent. The best is decided by the top performing model on the validation dataset which ran in less than one hour.

F STABILITY ACROSS REPLICATES

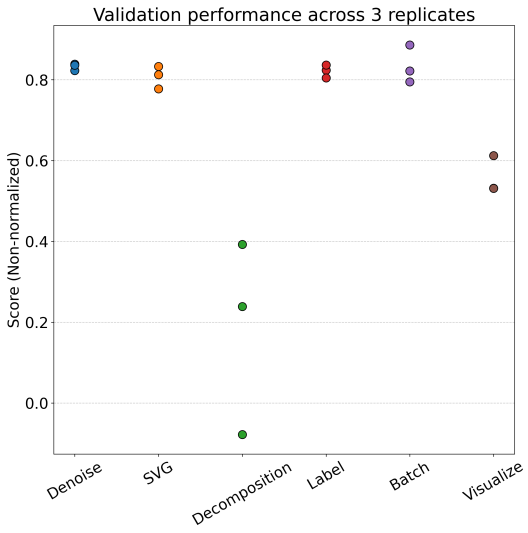

Figure 5: **Validation performance across 3 replicates.** Final validation performance after running TusoAI 3 separate times on each single-cell task.

G CODE DIVERSITY

We measure the diversity of generated code, as measured by the cosine similarity of the text embedding of one generated code versus all others. This is performed for TusoAI and AIDE. For each, we first filter out repetitive/uninformative code strings, including comments, imports, evaluation functions and data loading procedures (which will not change over iterations). We then apply sklearn's TfidfVectorizer function to each cleaned code to obtain a text embedding. We can then compute the cosine similarity between pairs of code. Diversity is measured as 1-cosine similarity. We opt for TF-IDF instead of more sophisticated methods like CodeBERT (Feng et al., 2020) which measure semantic similarity, as we observed this overestimated the similarity between code (all cosine similarity > 0.997 for all tasks). This is likely due to each iteration always being a slight permutation of the same python method performing the same task. TF-IDF better captures a measure of difference between algorithmic procedures in this case.

H ABLATION ANALYSIS

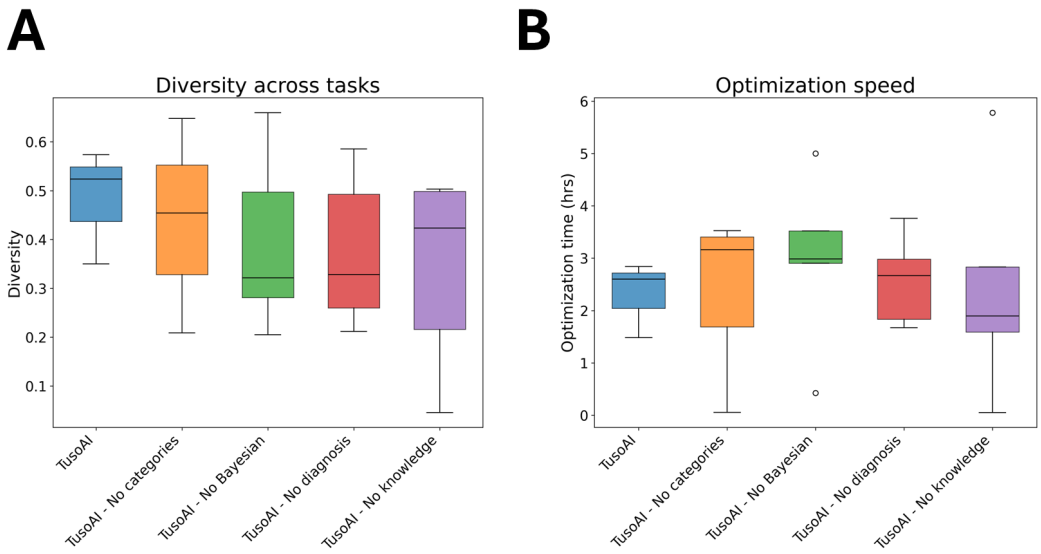

Figure 6: **Additional ablation information. (A)** Box plot across 5 tasks of the mean code diversity. **(B)** Box plot across 5 tasks of the mean time to optimize.

I LLM ANALYSIS

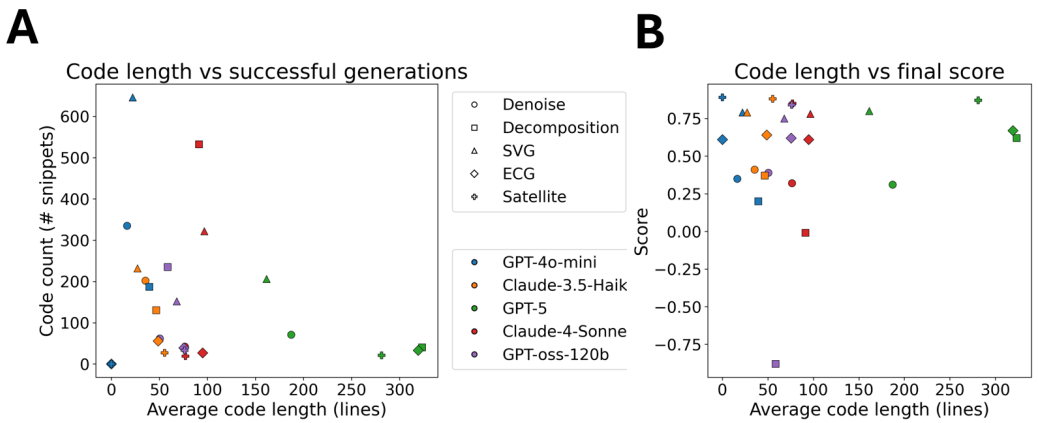

Figure 7: **Additional LLM information.** **(A)** Average length of generated methods for each task and LLM versus the total count of how many methods were generated. **(B)** Average length of generated methods for each task and LLM versus the final deployment performance.

J scDRS ANALYSIS

Optimization setup. scDRS' codebase consists of several files. We construct a version of compute_score.py that exposes the compute_raw_score function. This is the only function TusoAI operates upon during optimization. For optimization, we construct causal simulations similar to the scDRS paper, subsampling 10k cells from TMS, perturbing 1000 disease genes in a cluster of cells, setting the geneset overlap to 25%, and varying effect size from 5 to 50%. TusoAI optimizes the compute_raw_score function based on the average (F1 + AUPRC)/2 across 3 replicates at effect size 15%, where scDRS has lower power. We run this experiment for 24 hours using default parameter settings for TusoAI.

Additional simulation results. We apply scDRS and the learned version by TusoAI to all 30 replicates of each effect size in causal simulations. We additionally apply it to 100 replicates of null simulations, identical to scDRS, where 1000 random genes are selected with no perturbation. Additional metrics in these simulations are reported in Figure 8.

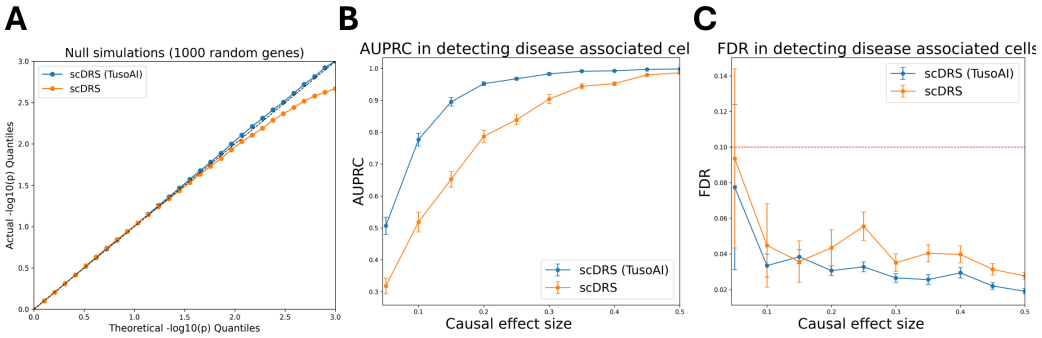

Figure 8: **Additional scDRS metrics.** **(A)** Q-Q plot of -log10 p-values in null simulations. 95% CI's are calculated at each point across 30 replicates. **(B)** AUPRC of associating individual cells in causal simulations. 95% CI's are calculated at each point across 30 replicates. **(C)** FDR of associating individual cells in causal simulations. 95% CI's are calculated at each point across 30 replicates.

K pgBOOST ANALYSIS

Optimization setup. pgBoost discovers that distance-based features are critical for modeling SNP-gene distances. It's samples are SNP-gene pairs, and features include over 30 features derived from single-cell multiome methods and 2 distance features, the SNP distance to the gene's transcription start site (TSS), and a binary indicator of if this is the closest TSS of any gene to the SNP. We augment pgBoost's script with gene annotations from GENCODE V48 (Mudge et al., 2025), specifically the SNP's position, the gene's TSS, and the gene's transcription end site (TES). During both the knowledge tree construction and optimization process, TusoAI is encouraged to come up with instructions/optimizations relevant to distance-based modeling of SNP-gene links and avoid other model changes. Optimization is performed by increasing the average enrichment in pgBoost's primary evaluation of gold-standard links (eQTL and ABC) relative to the original pgBoost's enrichment. We run TusoAI for 24 hours using default parameter settings.

L PROMPT TEMPLATES

L.1 INSTRUCTIONS FOR PARSING LITERATURE

Listing 5: Initializing paper description with abstract.

```
    prompt = f"""
You are a scientific summariser. Draft a concise yet technically accurate
description of the paper's method based **only** on the abstract below,
    to the extent possible. Capture the main points using bullets points.
     Do not waste words on complete sentences or details irrelevant to
    the methods.

Abstract:
\"\"\"{abstract}\"\"\"
"""
```

Listing 6: Updating paper description with methods section.

```
    prompt = f"""
The current method description ):

\"\"\"{current_desc}\"\"\"

New excerpt from the paper:
\"\"\"{new_text}\"\"\"

Update the description by **incorporating any new technical details or
    correcting
existing ones** found in the excerpt. Preserve conciseness and clarity.
    Return **only**
the revised description. Capture the main points using bullets points.
Do not waste words on complete sentences or details irrelevant to the
    methods.
Do not exceed {bp_limit} bullet points.
"""
```

L.2 INSTRUCTIONS FOR CONSTRUCTING CATEGORIES

Listing 7: Initializing categories with LLM.

```
    prompt = f"""
We are building an LLM-powered AutoML system for the task:

    "{task_description}"

    "{features_sentence}"

As a reference, some generic categories for optimizing classification
    models include:
{classification_categories}

You are a master of machine learning and the domain relevant to this task
    . Please first briefly reason about what kinds of modeling
    interventions or optimization strategies could be helpful for this
    specific task. Then propose a list of concise, task-relevant
    optimization categories.

Your list should include conceptual ideas that are tailored to this task
    and each should reflect a specific axis of improvement (e.g.,
    architectural choices, preprocessing tricks, domain constraints,
    evaluation metrics, robustness techniques, etc.).

Output exactly {num_cat} proposed categories, one per line, each enclosed
     in: <c>Category Name</c>

Do not include any other text, explanation, or formatting. By
    optimization we mean strictly performance, not runtime, scalability,
    logging, visualization, post-evaluation, etc. We will only have
    access to {data_available}.
"""
```

Listing 8: Updating categories with papers.

```
        prompt = f"""
We are building an LLM-powered AutoML system for the task:

    "{task_description}"

    "{features_sentence}"

We will curate and refine our categories based on the current categories
    and a paper.

Current categories:
{current}

Paper: "{title}"
Key method points:
{bullet_points}

TASK
1. If the paper suggests a *new* axis of optimization missing from the
    list,
   propose a concise category for it.
2. If two or more current categories can be merged, instead give a single
     name that
   subsumes them.
3. Otherwise, if the category is irrelevant given only {data_available},
    leave the list unchanged.

Return **one updated list only** one category per line,
```

```
each wrapped exactly like <c>Category</c>. No other text."""
```

L.3 INSTRUCTIONS FOR CONSTRUCTING WITHIN-CATEGORY INSTRUCTIONS

Listing 9: Initializing within-category instructions with LLM.

```
prompt = f"""
We are designing an LLM-powered AutoML system for the task:

    "{task_description}"

    "{features_sentence}"

Current optimisation axis: **{category}**

Below is a style example of prompts for a *regularisation* category for a
     classification task. Each prompt begins with *by ...* and expresses
    a specific, actionable optimisation idea:

{few_shot}

You are a master of machine learning and the domain relevant to this task
    . Keeping the same concise, actionable style, write **exactly {
    to_generate} distinct prompts** that belong to the **{category}**
    category **and are appropriate for this task**.

These should be a mix of general, conceptual, and complex prompts, and
    not overly specific, similar to the example.

Wrap *each* prompt in its own <p> ... </p> tag.
Return only these <p>...</p> lines, nothing else.
By optimization we mean strictly performance, not runtime, scalability,
    logging, visualizing, evaluating, etc. Assume the evaluation metrics
    already exist. We will only have access to {data_available}.
"""
```

Listing 10: Refining within-category instructions with LLM.

```
prompt = f"""
We are designing an LLM powered AutoML system for the task:

    "{task_description}"

We will only have access to {data_available}.

Here is a concise summary of the baseline method:
\"\"\"{summary}\"\"\"

Below are style examples of valid prompt lines taken from earlier work:
{few_shot_block}

Your job: generate between {n_new_min} and {n_new_max} new prompts. These
     will ultimately be assigned into one of the following categories:

{categories_line}

First, generate these prompts, independently of the categories. Second,
    assign each to it's most relevant category.
For each prompt output a line in this exact format:

<c>CategoryName</c><p>by    </p>

* Every prompt must begin with   by        .
* Cover a mix of general, conceptual, and complex ideas.
* Focus strictly on *performance* optimisation (ignore runtime,
    scalability, logging, etc.).
```

```
* Return only the '<c>   </c><p>_</p>' lines, nothing else.
""".strip()
```

L.4 INSTRUCTIONS FOR CONSTRUCTING INITIAL SOLUTIONS

Listing 11: Initializing solutions with LLM.

```
prompt = f"""
We are designing an LLM-powered AutoML system for the task:

    "{task_description}"

Below is an example list of generic model initializations for a **
    classification** task:
{few_shot}

You are a master of machine learning and of the domain relevant to this
    task.
Propose **exactly {num_init} concise model initializations** that could
    serve as
starting baselines **for this specific task** given that we only have
{data_available}. They should be general task-specific methods, model
    families, or high-level architectural
descriptions, not fully-specified pipelines.

Output one per line, each wrapped in <m> ... </m> tags.
Return *only* these <m>...</m> lines -- no explanations, no extra text.
"""
```

Listing 12: Refining initial solutions with LLM.

```
prompt = f"""
We are building an LLM-powered AutoML system for the task:

    "{task_description}"

We will curate and refine our *model initializations* list using insights
from the following paper.

Current initializations:
{current}

Paper: "{title}"
Key method points:
{bullet_points}

TASK ->
1. If the paper presents a **model family or architecture** not covered
    above,
   propose it as a concise initialization (<= 6 words).
2. If two or more current initializations are effectively the same family
    ,
   merge them by giving a single, clear name that subsumes them.
3. If the above are not met, or we cannot implement the model using {
    data_available}, leave the list unchanged.

Return **one updated list only** -- one initialization per line,
each wrapped exactly like <m>Initialization</m>.  No other text."""
```

L.5 PROMPT TEMPLATE FOR BIOMNI AND CHATGPT-AGENT

Listing 13: Single-cell denoising prompt template for scientific agents.

```
We are considering the task of single cell RNA-seq imputation.
We wish to create an expertly optimized model for this.
Here is a starter script. Create a top-performing model for our task
    within the tuso_model function.

import scanpy as sc
import pandas as pd
import numpy as np
import scipy as sp
import magic
from anndata import read_h5ad
import scprep
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
from scipy.sparse import issparse
from sklearn.decomposition import PCA
from anndata import AnnData
import random

def mse(adata):
    import anndata
    import scanpy as sc
    import scprep
    import sklearn.metrics

    test_data = anndata.AnnData(X=adata.obsm["test"], obs=adata.obs, var=
        adata.var)
    denoised_data = anndata.AnnData(
        X=adata.obsm["denoised"], obs=adata.obs, var=adata.var
    )

    # scaling and transformation
    target_sum = 10000

    sc.pp.normalize_total(test_data, target_sum=target_sum)
    sc.pp.log1p(test_data)

    sc.pp.normalize_total(denoised_data, target_sum=target_sum)
    sc.pp.log1p(denoised_data)

    error = sklearn.metrics.mean_squared_error(
        scprep.utils.toarray(test_data.X), denoised_data.X
    )
    return error
def tuso_model(adata):
    a = AnnData(
        X=adata.obsm["train"].copy(),
        obs=adata.obs.copy(),
        var=adata.var.copy()
    )

    out = a.X
    out = out.toarray() if issparse(out) else out
    adata.obsm["denoised"] = out
    return adata
def main():
    np.random.seed(42)
    random.seed(42)
    adata = read_h5ad('1k_pbmc_processed.h5ad')
    print("tuso_model_start")
```

```
    adata = tuso_model(adata)
    print("tuso_model_end")

    val_metric = 1-mse(adata)
    print(f"tuso_evaluate: {val_metric}")

main()



Make sure to store the denoised data in adata.obsm["denoised"].
Keep the function header, input, output the same.

Each time you generate code, run it, extract the tuso_evaluate metric,
    and try and build a better performing solution from the previous
    solutions.
```