TacitFlow: Learning Workflow Representations for Tacit-Knowledge-Grounded Machine Learning Engineering Agents

Yushan Jiang[1], Wenchao Yu[2†], Minyoung Choe[3], Dongjin Song[1†],
Jingchao Ni[4], Wei Cheng[2], Haifeng Chen[2]
[1]University of Connecticut [2]NEC Laboratories America [3]KAIST [4]University of Houston

Abstract

Large language model (LLM) agents have shown promise for automating machine learning engineering (MLE), but their iterative improvement remains weakly grounded. In particular, most existing methods lack structured representations of how expert pipeline components compose across tasks. Instead, their solution development is often driven by prior trials and flat textual context from external retrieval sources, with limited explicit support for reasoning over step dependencies and reusable pipeline compositions. To bridge this gap, we introduce TacitFlow, a knowledge-grounded MLE agent built on structured representations of expert workflows. We distill expert solutions across image, tabular, and text domains into heterogeneous workflow graphs of canonicalized pipeline steps and execution dependencies. From this graph, we learn contrastive node embeddings that capture multi-modal step semantics from technical descriptions and code snippets, and train a task-conditioned ranking model to capture workflow performance preferences from competition leaderboard rankings. During runtime, these representations support a retrieve, edit, and rerank refinement loop, where structured workflow context guides explicit candidate edits and the task-conditioned ranking model prioritizes promising workflows before execution. This design avoids exhaustively running every candidate while making agent decisions traceable to individual workflow steps. Experiments across three modalities show that our workflow-structured design yields promising performance on a diverse subset of competitions from MLE-Bench, while case studies illustrate transparent, workflow-grounded agent behavior on end-to-end MLE tasks.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; **Machine learning**.

Keywords

Representation Learning, Autonomous Data Science, Retrieval

1 Introduction

Building an effective machine learning solution for a new task is an iterative, knowledge-intensive process of exploring, evaluating, and adapting pipeline choices. Recent work on autonomous machine learning engineering (MLE) automates this process with LLM-based agents that propose, implement, and improve candidate solutions under a fixed budget [11, 26]. Despite promising progress, these agents continue to trail top human experts on real competitions [2], especially on tasks where strong solutions depend on tacit choices that are non-obvious from the data alone, such as cleaning suspicious labels or result calibration. Such choices accumulate in public competition notebooks written by expert practitioners. A natural question to ask is, *how can we represent tacit expert knowledge as meaningful, actionable guidance for constructing and improving MLE solutions?*

Existing MLE agents have made solution development more systematic by organizing the exploration space of candidate solutions. Beyond basic trial-and-error, recent systems use tree-structured search, evolutionary exploration, cross-branch reuse, or hierarchical caches to make exploration more efficient and targeted [3, 4, 13, 14, 21, 23, 26]. These methods structure how agents explore and refine their own generated solutions, but they do not provide expert knowledge about which pipeline choices tend to work. Recent work has begun exploring retrieval-augmented agents that bring in external materials such as web pages, papers, or curated domain notes to expose the LLM to external expertise [5, 16, 18]. Yet the retrieved knowledge is mostly represented as raw text summaries, case examples, or high-level hints. As a result, expert knowledge may influence the LLM through context, but it rarely participates in the agent's explicit decision process as a structured, reusable signal over pipelines that guides solution development.

To address this gap, we introduce TacitFlow, an MLE framework that exploits tacit expert knowledge as structured workflow guidance. Rather than using expert knowledge as passive context for proposing solutions, TacitFlow represents it as reusable pipeline steps, observed dependencies, and task-aware preferences over complete workflows. We first curate a heterogeneous workflow knowledge graph from 131 Kaggle competitions and 2,681 expert workflows across image, tabular, and text modalities. Each node corresponds to a canonical pipeline step with a technical name, natural-language notes, and code snippets, while each edge records an execution dependency observed in expert solutions, as shown in Figure 1. On top of this graph, we learn complementary signals at two granularities. At the node level, a self-supervised contrastive encoder produces step embeddings that capture both multi-modal semantic similarity and graph-structural proximity, yielding fine-grained step-level representations for retrieval. At the workflow level, a task-conditioned ranking model scores workflows by their expected quality on a given task. Supervised by competition leaderboard ranks, it provides a learned preference signal over candidate compositions. At runtime, TacitFlow guides the agent's next development step through a *retrieve-edit-rerank* loop. Given the target task and current solution state, the agent retrieves relevant steps via node embedding similarity, adapts them into candidate workflow changes, and uses the ranking model to select promising variants. The selected variants are then presented as suggested edits that condition the agent's next proposal. This efficiently makes expert

† Wenchao Yu and Dongjin Song are corresponding authors.

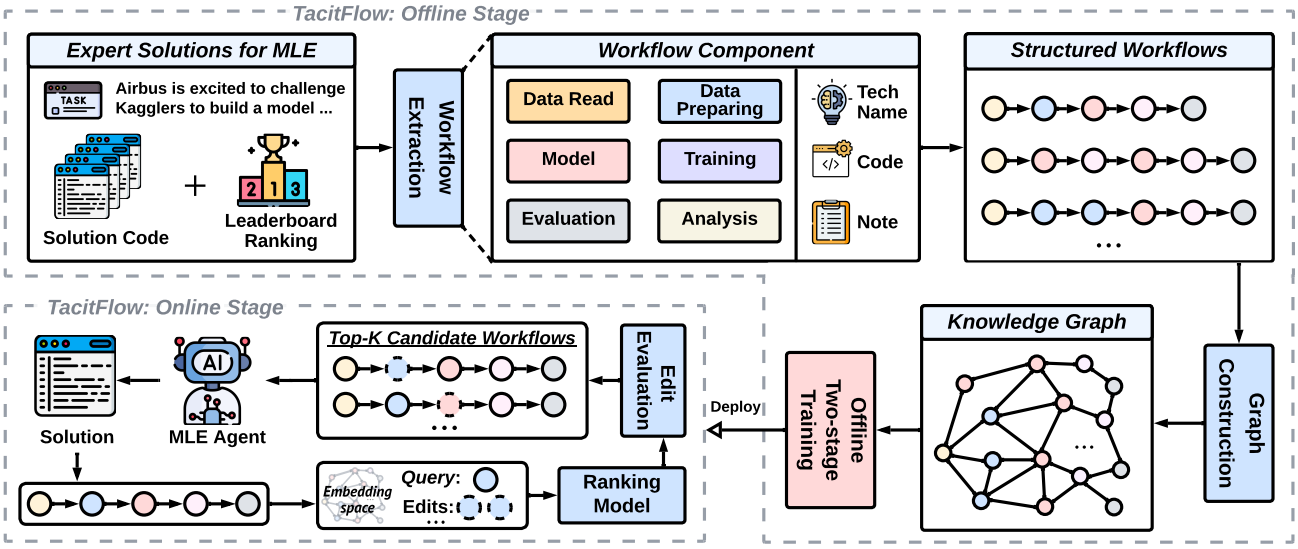

Figure 1: Overview of TacitFlow. The offline stage extracts structured workflows from expert MLE solutions, canonicalizes them into a workflow knowledge graph, and learns node embeddings together with a task-conditioned ranking model. The online stage uses these learned guidance signals to map the agent's current solution into a workflow, retrieve candidate edits, rerank edited workflows, and return selected workflow guidance to the MLE agent for refinement.

knowledge actionable without exhaustively testing every plausible modification. It also makes solution development more traceable, since each generated proposal can be linked to the retrieved steps attributed and prioritized by the rank. Our contributions are summarized as follows:

- We propose TacitFlow, a framework that turns tacit expert knowledge into a structured, learned representation for MLE agents, comprising reusable pipeline steps, execution dependencies, and task-aware quality preferences.
- We construct a heterogeneous workflow knowledge graph from 131 Kaggle competitions and 2,681 expert workflows across image, tabular, and text modalities, and propose bi-level representations over this graph to enable knowledge-grounded solution development.
- We integrate these learned representations into an iterative MLE agent through a retrieve-edit-rerank loop, making expert knowledge actionable and traceable while achieving performance gains on competitions across three modalities from MLE-Bench.

2 Related Work

LLM agents for machine learning engineering. Recent work on autonomous MLE formulates the problem as an iterative agent loop, with diverse systems advancing along two complementary axes: organizing the agent's own exploration of candidate solutions, and grounding agents in external knowledge [10, 19, 21, 26]. On the exploration axis, AIDE [11] initiates the paradigm with greedy tree search, AIRA [21] enhances this with structured operator sets and scoped memory across exploration branches, and the dominant subsequent line adopts Monte Carlo Tree Search and its variants over candidate solutions [3, 4, 18], including extensions with cross-branch reference edges and budget-aware planning. MLE-STAR [16] explores along a single chain with ablation-driven refinement, while another line pursues multi-agent evolutionary search across parallel populations [13, 23]. Memory mechanisms further enrich this axis: hierarchical cognitive caching of accumulated trials [14], and cross-branch trace reuse [3, 4] embed memory directly into the search loop. On the external knowledge axis, an early line of work uses curated case banks of expert solutions through case-based reasoning [7], while more recent agents bring expert materials into the LLM's context through web search [16], curated knowledge bases of papers and tricks [18] as well as playbooks and code repositories [15], domain priors over models, data, and strategies [4], and library documentation paired with execution histories [5]. Despite this expanding access, existing systems consume external knowledge as raw text, case examples, or condensed prompts, without building expert solutions into a learned representation that captures pipeline steps, dependencies, and task-conditioned preferences. TacitFlow departs from this by introducing such a representation and integrating it into the agent's proposal generation.

Learned representations of workflows. A growing line of work treats agentic workflows as graphs and learns representations over them. Workflow optimization methods [9, 27, 30] formulate workflow design as a search problem over communication topologies or code-represented workflows, typically optimized with reinforcement learning, meta-agent programming, or Monte Carlo Tree Search. Building on this paradigm, recent work has begun to learn binary performance predictors over agentic workflows so that promising candidates can be prioritized without expensive execution. FLORA-Bench [28] releases 600k workflow-task pairs over multi-agent communication topologies and trains graph-based predictors of workflow success on short reasoning and code-completion benchmarks. Furthermore, Agentic Predictor [22] adds code- and prompt-aware encoders, and GLOW [6] integrates LLM-based reasoning into graph predictors. We extend this line of work

into the substantially more demanding setting of MLE tasks. Compared to short reasoning or code-completion tasks solved by homogeneous agent communication, ML pipelines exhibit a heterogeneous and multi-stage compositional structure, including preprocessing, modeling, and training choices that interact in task-dependent ways, and competition leaderboards provide dense, fine-grained quality signals over complete workflows, going beyond binary success labels. Moreover, TacitFlow shifts learned workflow representations from passive performance estimators to active components of an MLE agent's decision process, reasoning over top candidate edits at each refinement step.

3 Problem Formulation

We consider the autonomous machine learning engineering task, where an agent produces a high-quality solution for a given ML competition under a fixed time budget. Formally, given an ML task with an associated dataset $\mathcal{D} = \{\mathcal{D}_{\text{dev}}, \mathcal{D}_{\text{test}}\}$, the agent iteratively develops candidate solutions s using only $\mathcal{D}_{\text{dev}}$, while final performance is measured by a task-specific metric $M(s; \mathcal{D}_{\text{test}})$ on the held-out test set. Given a budget vector B (*e.g.,* time and token usage), the agent's goal is to solve the constrained optimization as:

$$s^\star = \arg\max_{s \in \mathcal{S}} \; M(s; \mathcal{D}_{\text{test}}), \quad \text{s.t. budget}(s) \le B \tag{1}$$

where $\mathcal{S}$ is the feasible solution space and time(s) is the total elapsed time for producing s.

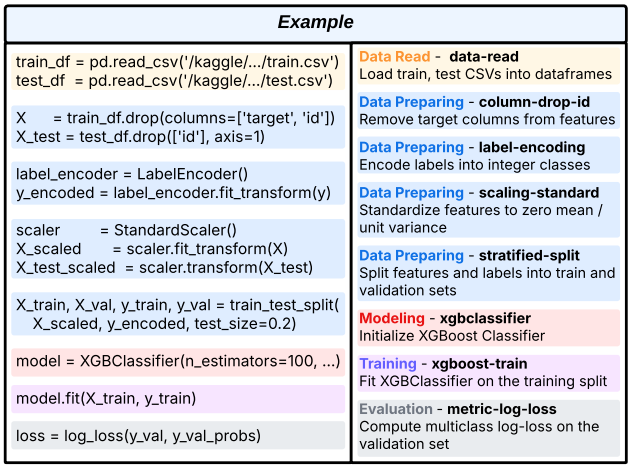

Figure 2: An example of workflow extraction from a code solution.

4 Methodology

We propose a knowledge-grounded approach that equips the agent with structured workflow knowledge distilled from similar competition notebooks. For each data modality (*e.g.*, image, tabular, text), we use an LLM-based extractor to convert top-performing solutions into evidential workflow steps that capture canonical operations, functional roles, and supporting code context, as shown in Figure 2. These steps are organized into a heterogeneous knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where recurring operations across solutions are consolidated into shared nodes and observed execution dependencies define edges. The resulting graph exposes reusable workflow structure for step-level retrieval and workflow-level ranking. A complete workflow w thus corresponds to a subgraph paired with a task description drawn from the competition metadata. The details are provided in Appendix B. Over this graph, we learn complementary signals at two granularities: node embeddings $\mathbf{h}_v$ that capture multi-modal semantic similarity and graph-structural proximity, and a task-conditioned workflow ranking model that scores complete workflows by empirical quality. These bi-level representations serve as structured context for an iterative MLE agent, shifting solution development toward a knowledge-guided process that surfaces expert-derived candidates at every step and yields transparent decision traces.

4.1 Learning Node Representations

Multi-modal Graph Encoder. We build a graph encoder $f_\theta : \mathcal{G} \to \mathbb{R}^{|\mathcal{V}| \times d}$ with hidden dimension d. For each feature modality $k \in$ cat, tech, notes, code (category label, technique name, technique notes, and code snippets), a projection ϕ_k maps raw features to a modality-specific subspace with dimension d'_k chosen per modality (one-hot for cat, S-BERT [20] for tech, notes and CodeT5+ [24] for code). The projected features are concatenated and fused via a two-layer MLP:

$$\mathbf{h}_v^k = \phi_k(\mathbf{x}_v^k), \tilde{\mathbf{h}}_v = \text{MLP}([\mathbf{h}_v^{\text{cat}} \| \mathbf{h}_v^{\text{tech}} \| \mathbf{h}_v^{\text{notes}} \| \mathbf{h}_v^{\text{code}}])$$

The graph encoder consists of L GraphSAGE layers [8] and residual connections:

$$\mathbf{h}_v^{(\ell+1)} = \mathbf{h}_v^{(\ell)} + \text{GELU}\Big(\text{SAGE}\big(\mathbf{h}_v^{(\ell)}, \{\mathbf{h}_u^{(\ell)}\}_{u \in \mathcal{N}(v)}\big)\Big)$$

Let $\mathbf{h}_v = f_\theta(\mathcal{G})_v$ denote the encoder output after L layers. For contrastive pretraining, we attach a two-layer projection head g and obtain the contrastive embedding $\mathbf{z}_v = g(\mathbf{h}_v)$. Unless otherwise specified, we use $\mathbf{h}_v$ (before g) as the final node representation for downstream usage.

We pretrain node embeddings to distill these cross-workflow patterns together with each node's multi-modal attributes via self-supervised contrastive learning, applying graph augmentations along edge perturbations and feature masking [29]. Specifically, two correlated graph views $\tilde{\mathcal{G}}_1, \tilde{\mathcal{G}}_2$ are generated by independently applying *(i)* stochastic edge dropping with rate p_e and *(ii)* feature-dimension masking with rate p_f. The two views are processed by the shared encoder and projection head, yielding $\mathbf{z}_v^1 = g\big(f_\theta(\tilde{\mathcal{G}}_1)_v\big)$ and $\mathbf{z}_v^2 = g\big(f_\theta(\tilde{\mathcal{G}}_2)_v\big)$ with dimension d.

Cross-view contrastive loss. We first maximize agreement between representations of the same node across views via a symmetric InfoNCE objective [17]:

$$\mathcal{L}_{\text{cl}} = -\frac{1}{2N} \sum_{v \in \mathcal{V}} \sum_{a \neq b} \log \frac{\exp\big(\text{sim}(\mathbf{z}_v^a, \mathbf{z}_v^b)/\tau\big)}{\sum_{v' \in \mathcal{V}} \exp\big(\text{sim}(\mathbf{z}_v^a, \mathbf{z}_{v'}^b)/\tau\big)}$$

where $a, b \in \{1, 2\}$ and $a \neq b$ indicate two views, $\text{sim}(\cdot, \cdot)$ is cosine similarity and τ is a temperature parameter.

VICReg regularization. To prevent dimensional collapse of the learned representations [1], we add variance and covariance style regularization terms on both views. The variance term $\mathcal{L}_{var}$ maintains per-dimension standard deviation, while the covariance term $\mathcal{L}_{cov}$ penalizes off-diagonal entries of the covariance matrix C^i to

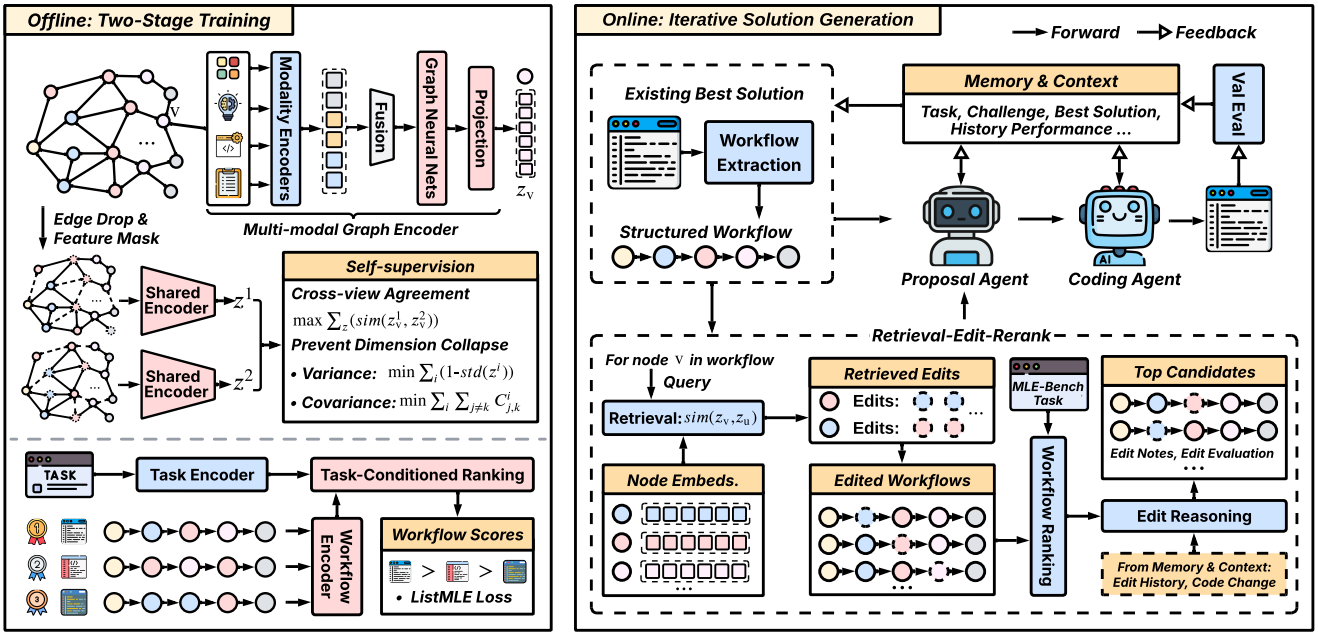

Figure 3: Methodology of TacitFlow. Stage 1 learns multi-modal node embeddings over a heterogeneous workflow graph with self-supervised objectives. Stage 2 learns a task-conditioned ranking model over workflow representations to score complete workflows by expected quality. Stage 3 converts the current best solution into a structured workflow, retrieves candidate edits for a new task using node embeddings, reranks them with the ranking model, and refines the solution through a proposal agent under validation feedback and memory.

decorrelate dimensions:

$$\mathcal{L}_{\text{var}} = \frac{1}{2d}\sum_{i,j}\max\big(0, 1 - \text{std}(\mathbf{z}^i_{:,j})\big), \quad \mathcal{L}_{\text{cov}} = \frac{1}{2d}\sum_{i,j\neq k}(C^i_{jk})^2$$

The full training objective is:

$$\mathcal{L} = \lambda_{\text{cl}}\,\mathcal{L}_{\text{cl}} + \lambda_{\text{var}}\,\mathcal{L}_{\text{var}} + \lambda_{\text{cov}}\,\mathcal{L}_{\text{cov}}. \tag{2}$$

4.2 Task-Conditioned Workflow Ranking

The node embeddings encode rich multi-modal semantics and graph-aware structural patterns, making them well suited for workflow level exploitation. To retain these properties, we fix the node embeddings and cast workflow selection as a ranking task within each competition.

Let $\mathbf{X}_w \in \mathbb{R}^{n\times d}$ denote the sequence of node embeddings (padded to length n) and $\mathbf{M}_w \in \{0,1\}^n$ the binary mask of valid positions. Task descriptions t_w (from the competition metadata) are encoded with Sentence-BERT and projected by an MLP as $\mathbf{t}'_w$. In this stage, our goal is to learn a task-conditioned scoring function s_w that reflects the relative quality of workflows from the same competition. To this end, we employ a lightweight sequence encoder to obtain a workflow representation and then fuse it with the task embedding to predict a preference score.

Workflow Encoder. Because workflow steps execute sequentially, we encode the step sequence with a unidirectional LSTM that represents the execution dependencies: $\mathbf{H}_w = \text{LSTM}(\mathbf{X}_w) \in \mathbb{R}^{n\times d_h}$, where each hidden state $\mathbf{h}^{\text{seq}}_i$ summarizes the workflow up to step i. To highlight which steps matter most, we apply an attentive pooling mechanism that scores each hidden state against a single learned query vector $\mathbf{q} \in \mathbb{R}^{d_h}$:

$$\alpha_i = \frac{\mathbf{M}_{w,i}\,\exp(\mathbf{q}^\top \mathbf{h}^{\text{seq}}_i)}{\sum_j \mathbf{M}_{w,j}\,\exp(\mathbf{q}^\top \mathbf{h}^{\text{seq}}_j)}, \quad \mathbf{h}_w = \sum_i \alpha_i\,\mathbf{h}^{\text{seq}}_i$$

where the mask $\mathbf{M}_w$ zeros out padded positions to handle varying workflow lengths. The attention weights α_i serve as interpretable step-wise importance scores. We fuse the workflow and task representations via an MLP ϕ_{f} and predict a scalar score as: $s_w = \phi_{\text{f}}([\mathbf{t}' \| \mathbf{h}_w])$.

Ranking Loss. To supervise workflow ranking, we use the public leaderboard ranking of each competition as listwise supervision, yielding a ground-truth ranked list σ_c over the n_c workflows in competition c, ordered from best to worst. We adopt the ListMLE loss [25], a listwise objective that models the likelihood of the entire permutation under a Plackett-Luce model:

$$\mathcal{L}_{\text{rank}} = -\frac{1}{|C|}\sum_{c\in C}\sum_{j=1}^{n_c}\log\frac{\exp(s_{w_{\sigma_c(j)}})}{\sum_{\ell=j}^{n_c}\exp(s_{w_{\sigma_c(\ell)}})} \tag{3}$$

where C is the set of all competitions. This encourages the model to score workflows consistently with the ground-truth order across the entire candidate set.

4.3 TacitFlow-Informed MLE Agent

The workflow representations encode reusable knowledge about which solution patterns tend to perform well and how pipeline components relate structurally, which we seamlessly integrate into a solution development loop while maintaining transparent decision

traces. As our starting point, we adopt a base loop with components common to recent open-source MLE agents [11, 14, 18, 26]: tree-structured search that selectively expands promising branches, an exploration memory of past solutions and validation feedback, and stage-aware prompting strategies, and a standard coding workflow that implements proposals into runnable code via subset-based prototyping [26].

Within each search branch, we consider an MLE agent that alternates between proposing ideas, generating and evaluating code, and updating an exploration history $\mathcal{H}_{<t}$. Let $\hat{c}_{t-1}$ be the latest accepted solution before iteration t, and κ_t be the structured context derived from workflow representations. The agent performs:

$$i_t \leftarrow \textsc{Propose}(\kappa_t,\ \mathcal{H}_{<t})$$
$$c_t \leftarrow \textsc{Implement}(i_t;\ \hat{c}_{t-1})$$
$$m_t \leftarrow \textsc{Evaluate}(c_t; \mathcal{D}_{\text{dev}})$$

where i_t is the initial/refinement proposal, c_t the resulting code, and m_t the validation performance. If c_t yields an improvement in m_t over the current baseline, we accept it as the new solution $\hat{c}_t$; otherwise, we keep the baseline unchanged. The validation outcome and associated feedback, including execution status, error and debugging traces, and summaries of historical solutions, are appended to the history $\mathcal{H}_{<t}$ to inform subsequent iterations.

Without external structure, an iterative MLE agent must propose refinements based solely on its parametric knowledge and accumulated trial history. While the history records past attempts and outcomes, it does not explicitly expose which workflow components are most promising to modify, nor does it provide a principled mechanism for reusing high-quality solution patterns observed in related tasks. To enable structured and traceable reasoning over solution components, we first convert the current solution code into the same workflow abstraction used in the offline stage:

$$w_{t-1} = \textsc{Extract}(\hat{c}_{t-1}) = [v_0, v_1, \cdots, v_n],$$

where Extract denotes an LLM-based procedure that decomposes code into step-level components. This structured representation provides an interpretable view of the solution and allows subsequent refinement decisions to be attributed to specific steps.

Step-level retrieval. We first map each step in the current workflow w_{t-1} into the pretrained node embedding space. As the agent may produce steps that do not appear verbatim in the offline graph $\mathcal{G}$, we employ a three-tier matching scheme to bridge the gap between runtime workflows and the offline corpus: *(i)* Exact match: if the step's standardized name appears in $\mathcal{V}$, we reuse the corresponding embedding $\mathbf{z}_v$. *(ii)* Fuzzy match: when no exact match exists, we check for substring containment, and use S-BERT cosine similarity, assigning the embedding of the closest matched node. *(iii)* Constructed embedding: when no sufficiently similar entry exists, we construct an embedding by feeding the step's multi-modal features via the pretrained projection ϕ_k. Given these embeddings, we retrieve semantically related alternatives for each step $v \in \mathcal{V}_{w_{t-1}}$:

$$\mathcal{N}_{\text{ref}}(v) = \{\, u \in \mathcal{V} : \text{sim}(\mathbf{z}_v, \mathbf{z}_u) \geq \delta,\ u \neq v \,\}$$

where $\mathbf{z}_v$ denotes the pretrained node embedding. The retrieved nodes serve as candidate substitutions grounded in multi-modal semantic and structural similarity learned via graph pretraining.

Edit and Rerank. Each retrieved node $u \in \mathcal{N}_{\text{ref}}(v)$ is treated as a candidate workflow edit. If u shares the same stage category as the original step v, it is used as an in-place substitute; otherwise, it suggests a cross-stage edit. Each candidate workflow $\tilde{w}$ is generated by applying a single edit to w_{t-1}, and inherits the stage category of the edited step. We score each candidate with our task-conditioned workflow ranker $s_t(\tilde{w})$, and select the top candidates per stage category, preventing single-stage dominance and exposing the agent to refinements across the full pipeline:

$$\mathcal{W}_t^\star = \bigcup_c \text{top-}k\{s_t(\tilde{w}) : \text{cat}(\tilde{w}) = c\}$$

The shortlist $\mathcal{W}_t^\star$ serves as the basis for the structured knowledge context κ_t, further enriched in the next step.

Edit Reasoning and Proposal. $\mathcal{W}_t^\star$ is task-conditioned but does not reflect the agent's recent execution experience, namely, which edits have been tried, which succeeded or failed, and how the validation score has evolved. We therefore maintain a lightweight memory to inform reasoning over retrieved edits. At each iteration t, we enrich $\mathcal{W}_t^\star$ by conditioning each retrieved edit on the task context and accumulated lessons, attaching a short advisory note that identifies recent conflicts, suggests compatible edits, and highlights task-specific constraints. Informed by these notes and the current solution state, the proposal agent classifies each edit as adopt, consider, reject, or irrelevant, and organizes the resulting decisions and rationales into the structured knowledge context κ_t used to draft each proposal. It then drafts several candidate proposals, scores them, and either selects one or ensembles multiple candidates into the final proposal i_t passed to the coding agent. After execution, the proposal, code change, historical edits, and validation outcome are summarized and appended to memory, where they can inform edit reasoning in subsequent iterations. The complete procedure is summarized in Algorithm 1 (Appendix A).

5 Experiments

5.1 Experiment Setup

Datasets, Baselines, and Metrics. We evaluate TacitFlow on a diverse 18-competition subset of MLE-Bench [2] spanning three input modalities (image, tabular, text), three difficulty tiers (easy, medium, hard), and diverse task types. For each competition we report the native evaluation score and the highest medal threshold attained (Above Median, Bronze, Silver, Gold), averaged over three runs. We compare against six open-source MLE agents spanning dominant design axes: AIDE [11], ML-Master [14], AIRA [21], R&D-Agent [26], Leeroo (KAPSO) [15], and MLEvolve. The baseline results are averaged from their submissions to the official MLE-Bench GitHub repository.

Environment and LLM Configurations. Each run uses a 12-hour wall-clock budget on a server with NVIDIA Quadro RTX 6000 (24 GB), 48 CPU cores, and 256 GB RAM. For runtime MLE we use `gpt-5.1` with reasoning disabled (*none* effort). Offline workflow graph construction uses `gpt-4.1` and `gpt-4o-mini` for workflow extraction, for canonicalization, and code feature cleansing.

Workflow Graph. The workflow graph supporting both offline representation learning and online edit retrieval is built from 131

Table 1: Per-competition results on an 18-competition subset of MLE-bench. The mean of valid submission scores across that agent's runs are reported. Medal coloring derived from the mean: ● gold, ● silver, ● bronze, ● above median, plain text otherwise. "–" = no valid submission across runs. * indicates RD-Agent fallback results obtained with o3+GPT-4.1, used when the GPT-5 run produced no valid submission. ↑ higher is better, ↓ lower is better.

	Task	Competition	Metric	AIDE	ML-Master	AIRA	RD-Agent	Leeroo	MLEvolve	TacitFlow
Image	Classification	dog-breed	LogLoss↓	2.749	● 0.388	0.861	● 0.372*	–	● 0.379	● 0.364
		plant-pathology-20	mROC-AUC↑	● 0.977	● 0.989	● 0.968	● 0.992	● 0.996	● 0.997	● 0.994
		dogs-vs-cats	LogLoss↓	1.769	● 0.004	0.162	● 0.027	● 0.004	● 0.004	● 0.009
	Segmentation	tgs-salt	mAP@IoU↑	0.394	0.252	0.643	0.793*	–	● 0.836	● 0.824
		uw-gi-tract	Dice+Haus↑	0.186	0.050	0.430	0.408*	–	0.802	0.571
	Detection	kuzushiji	F1 Score↑	0.141	0.410	0.345	0.316	● 0.661	● 0.909	● 0.611
	Regression	denoising-dirty	RMSE↓	0.082	● 0.011	● 0.047	● 0.011	● 0.031	● 0.008	● 0.011
Text	Classification	spooky-author	LogLoss↓	0.489	● 0.279	● 0.247	● 0.259	● 0.254	● 0.214	● 0.301
		detect-insults	AUC↑	● 0.905	● 0.922	● 0.949	● 0.928	● 0.904	● 0.954	● 0.945
		jigsaw-toxic	mROC-AUC↑	0.970	● 0.983	● 0.986	● 0.986	● 0.987	● 0.988	● 0.984
	Regression	us-patent	Pearson↑	0.718	0.849	● 0.860	0.801	● 0.870	● 0.865	0.818
		essay-score-2	QWK↑	0.737	0.821	0.826	0.825	● 0.859	● 0.838	● 0.830
	Seq2Seq	text-norm-russian	Accuracy↑	0.927	0.963	0.968	● 0.981	● 0.976	● 0.989	● 0.978
Tabular	Classification	random-acts	AUC↑	● 0.643	● 0.607	● 0.629	● 0.786	–	● 0.700	● 0.797
		tps-dec-2021	Accuracy↑	0.928	● 0.962	● 0.954	● 0.963	● 0.963	● 0.962	● 0.963
	Regression	nomad2018	mRMSLE↓	0.102	● 0.063	0.085	● 0.060	● 0.062	● 0.059	● 0.059
		tps-may-2022	AUC↑	0.914	● 0.992	● 0.990	0.970*	–	● 0.998	● 0.979
		stanford-vaccine	MCRMSE↓	0.465	● 0.247	0.461	● 0.269	● 0.333	● 0.214	● 0.236
	LLM Model & Configuration			o1-preview	deepseek-r1	o3	GPT-5	Gemini-3-Pro-Preview	Gemini-3-Pro-Preview	GPT-5.1 (No Reasoning)
	Execution Time (Hours)			24	12	24	12	24	12	12

Table 2: Summary statistics of the collected workflow corpus.

Statistic	Image	Tabular	Text
Competitions	48	55	28
Workflows	883	1,226	572
Nodes	3,536	3,836	2,754
Edges	10,356	14,510	8,054
Avg. edges / node	2.93	3.78	2.92
Mean workflows / competition	18.4	22.3	20.4

historical Kaggle competitions and 2,681 expert workflows. Summary statistics of the corpus, including modality and stage-category coverage, average workflow length, and node degree distributions are provided in Table 2. Appendix B provides the list of competitions and details of workflow collection, extraction, graph construction, canonicalization, deduplication, and code quality control.

Leakage Control. Before crawling notebooks, we removed all competitions appearing in MLE-Bench using official competition IDs and manually checked aliases. During evaluation, the agent has no internet access and cannot retrieve target-competition notebooks, discussions, or solutions. The workflow graph and ranker supervision are constructed only from competitions disjoint from the 18 evaluation competitions listed in Appendix Table 8.

Hyperparameters. For offline training, we use Optuna for hyperparameter selection, using Adam [12] as the optimizer. For online MLE, we use two branches in Monte Carlo Tree Search, and select up to 3 top candidates per category in our retrieve-edit-rerank loop.

5.2 Runtime Experiments

Table 1 reports per-competition results on the 18-competition subset of MLE-Bench. TacitFlow uses gpt-5.1 with reasoning effort set to *none*, whereas most baselines rely on reasoning models such as o1-preview, DeepSeek-R1, o3, or Gemini-3-Pro-Preview. Despite using this non-reasoning configuration, TacitFlow achieves the best score on 4 competitions and the second-best score on 5, reaches at least above-median performance on 16 of 18 competitions, and produces valid submissions for all 18. Among matched-budget systems with a 12-hour wall-clock limit, TacitFlow matches or exceeds R&D-Agent (GPT-5) on a majority of competitions. In contrast, Leeroo runs with a 24-hour budget using Gemini-3-Pro-Preview but fails to produce valid submissions on 5 of 18 competitions, including 4 medium-complexity tasks (`tgs-salt`, `uw-gi-tract`, `random-acts`, and `tps-may-2022`). This highlights a robustness gap beyond aggregated medal rates. MLEvolve is the best baseline in the current MLE-bench open-source leaderboard. The gap to TacitFlow concentrates on tasks demanding specialized architectural exploration, notably segmentation and detection (`uw-gi-tract`, `kuzushiji`) where MLEvolve's reasoning model is most beneficial. On tabular tasks, where solution quality is dominated by reusable feature engineering and modeling patterns rather than open-ended reasoning, TacitFlow reaches the best score on 3 of 4 competitions (`random-acts`, `tps-dec-2021`, `nomad2018`), matching or surpassing MLEvolve despite using a non-reasoning LLM. We read this as evidence that explicit, learned workflow representations are a

complementary signal that can recover much of the gap from a non-reasoning backbone.

5.3 Ablation Study

The effectiveness of the retrieve-edit-rerank design. Table 3 evaluates the retrieve-edit-rerank design on five medium- and hard-complexity MLE-Bench competitions. We compare TacitFlow with a *Vanilla* variant that uses the same iterative MLE scaffold but disables the workflow knowledge graph, and report both early progress and final performance. Across these challenging tasks, TacitFlow improves final performance on all five competitions, with an average improvement of 45.4%. This suggests that retrieved expert workflow components provide useful guidance when strong solutions require nontrivial pipeline choices. The same design also improves early-stage robustness: TacitFlow reaches the first valid submission faster on four of the five competitions and reduces the average first-valid time from 6.86 to 5.73 hours, corresponding to a 14.1% improvement. These gains indicate that retrieve-edit-rerank benefits both dimensions of the agent loop: it helps the agent reach executable solutions sooner and steers subsequent refinement toward higher-performing workflow variants, rather than spending the budget rediscovering common pipeline patterns through less-focused trial and error.

Table 3: Effect of the retrieve-edit-rerank design on early progress and final performance across five medium and hard competitions. First-valid denotes the time to obtain the first valid submission.

Competition	First-valid Time (h)			Performance		
	Ours	Vanilla	Δ	Ours	Vanilla	Δ
kuzushiji	7.92	9.34	-1.42	0.611	0.220	+0.391
tgs-salt	4.29	4.66	-0.37	0.824	0.761	+0.063
uw-gi-tract	5.65	6.41	-0.76	0.523	0.416	+0.107
stanford-vaccine	3.91	4.18	-0.17	↓0.236	↓0.268	+0.032
essay-score-2	6.88	9.72	-2.84	0.830	0.802	+0.028
Average Imp.	14.1% (5.73 vs 6.86)			45.4%		

Component ablation of learned guidance. We further ablate the representation learning and ranking components that provide learned guidance in TacitFlow. We use two complementary evaluations. The first is an in-corpus evaluation, which performs cross-validation over the expert workflow corpus used to build the workflow graph. The second is an out-of-corpus evaluation on solution trajectories from our TacitFlow MLE-Bench runs, where each intermediate solution is converted into the same workflow abstraction and ranked by the learned model. In this setting, the ground-truth rank is determined by the official MLE-Bench score of each submission, providing a direct test of whether the learned model captures true competition-level workflow quality. Tables 4 and 5 evaluate whether the learned guidance model recovers the true quality ordering of workflows. We report NDCG@5, MRR, and Top-3 accuracy to measure top-ranked workflow quality, the rank of the best workflow, and whether it appears among the top candidates. Across both in-corpus and out-of-corpus evaluations, the full model achieves the strongest overall performance among all variants, showing

Table 4: Model ablation (in-corpus, 5-fold CV × 3 seeds).

	Condition	NDCG@5	MRR	Top-3
Image	w/o Pretrain.	0.888 ± 0.032	0.449 ± 0.133	0.584 ± 0.195
	w/o VICReg.	0.880 ± 0.026	0.450 ± 0.118	0.574 ± 0.128
	w/o Attn.	0.889 ± 0.022	0.459 ± 0.122	0.575 ± 0.159
	Ours	**0.897 ± 0.037**	**0.488 ± 0.132**	**0.612 ± 0.180**
Text	w/o Pretrain.	0.786 ± 0.061	0.315 ± 0.110	0.324 ± 0.181
	w/o VICReg.	0.804 ± 0.067	0.427 ± 0.073	0.522 ± 0.135
	w/o Attn.	0.826 ± 0.060	0.407 ± 0.104	**0.547 ± 0.181**
	Ours	**0.833 ± 0.051**	**0.434 ± 0.139**	0.529 ± 0.258
Tabular	w/o Pretrain.	0.775 ± 0.093	0.493 ± 0.150	0.673 ± 0.231
	w/o VICReg.	0.795 ± 0.055	0.534 ± 0.113	0.709 ± 0.204
	w/o Attn.	0.769 ± 0.079	0.475 ± 0.166	0.636 ± 0.220
	Ours	**0.814 ± 0.058**	**0.540 ± 0.109**	**0.720 ± 0.215**

Table 5: Model ablation (out-of-corpus evaluation on MLE-Bench runs, 3 seeds).

	Condition	NDCG@5	MRR	Top-3
Image	w/o Pretrain.	0.645 ± 0.019	0.258 ± 0.012	0.202 ± 0.110
	w/o VICReg.	0.650 ± 0.002	0.256 ± 0.008	0.260 ± 0.015
	w/o Attn.	0.675 ± 0.022	0.242 ± 0.008	0.271 ± 0.059
	Ours	**0.688 ± 0.016**	**0.284 ± 0.025**	**0.344 ± 0.026**
Text	w/o Pretrain.	0.767 ± 0.021	0.315 ± 0.025	0.508 ± 0.011
	w/o VICReg.	0.746 ± 0.025	0.391 ± 0.020	0.413 ± 0.040
	w/o Attn.	0.724 ± 0.017	0.389 ± 0.028	0.421 ± 0.045
	Ours	**0.783 ± 0.023**	**0.519 ± 0.008**	**0.587 ± 0.092**
Tabular	w/o Pretrain.	0.561 ± 0.013	0.178 ± 0.020	0.083 ± 0.059
	w/o VICReg.	0.567 ± 0.042	0.194 ± 0.012	0.208 ± 0.059
	w/o Attn.	0.638 ± 0.019	0.281 ± 0.027	**0.375 ± 0.000**
	Ours	**0.650 ± 0.024**	**0.433 ± 0.023**	**0.375 ± 0.000**

that the learned representations provide a useful workflow preference signal. Contrastive pretraining with variance and covariance regularization brings consistent gains, suggesting the importance of robust self-supervised step representations that capture both semantic similarity and structural neighborhoods. Step-level attention further improves ranking by adaptively weighting workflow components according to their relevance to solution quality.

On the other hand, the transition from in-corpus evaluation to MLE-Bench trajectory evaluation introduces a moderate performance drop, reflecting the distribution shift from curated expert workflows to unseen agent-generated solutions. Nevertheless, the learned workflow guidance remains effective under this harder out-of-corpus setting, capturing useful ranking signals that generalize beyond the expert workflow corpus. We provide the full results with more metrics (NDCG@10, Top1) in Table 12 and 13, as well as other ablations including the feature importance (technique name, code, notes) across three image, text, and tabular modalities, in Table 9, 10, 11 of Appendix B.5.

5.4 Case Study: Traceable Refinement

We provide a case study that illustrates a single refinement step on `plant-pathology-2020-fgvc7`, as shown in Figure 4. Starting from the agent's current solution, TacitFlow first extracts a 21-step workflow spanning data preparation, modeling, training, and

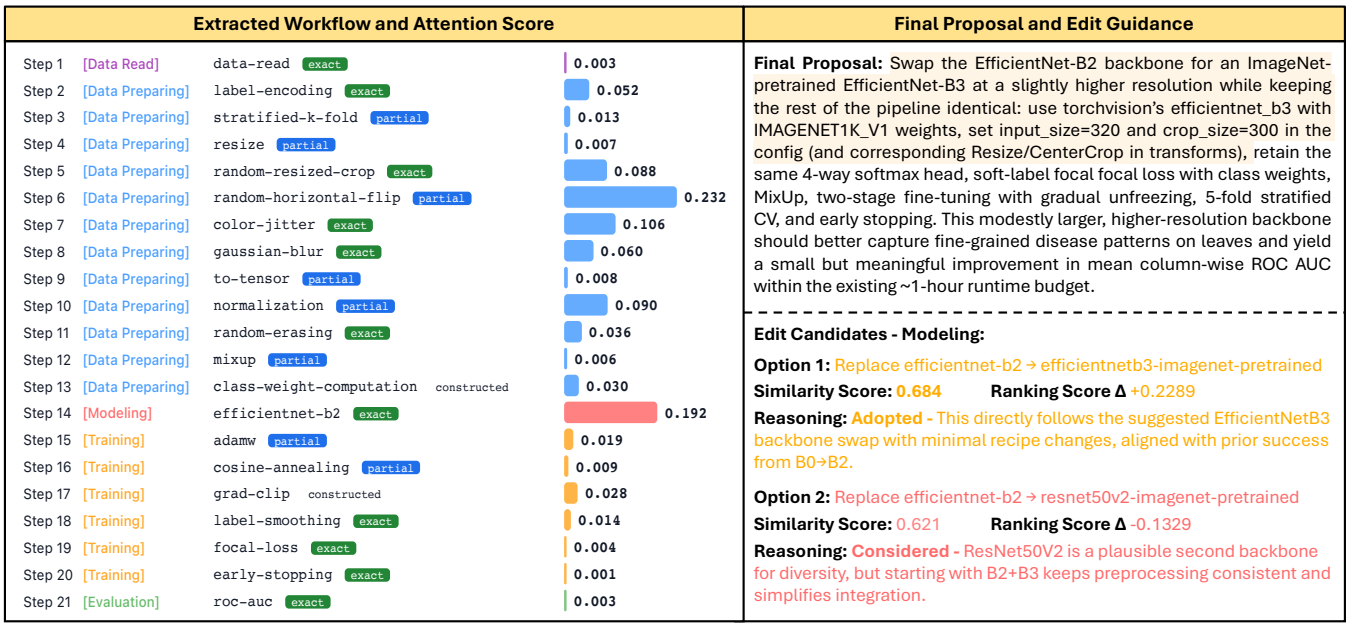

Figure 4: Case study on *plant-pathology-2020-fgvc7*. Left: the current solution is represented as an extracted workflow with step-level attention scores. Right: TacitFlow retrieves and ranks candidate modeling edits, then forms the final proposal by adopting the EfficientNet-B3 backbone upgrade.

evaluation. Each step is represented by a stage category, canonical technique name, matching status, and an attention score from the workflow ranker. The matching status records how the runtime step is connected to the offline workflow graph: common operations such as `data-read`, `label-encoding`, and `efficientnet-b2` are matched exactly, while several augmentation and training steps are partially matched or constructed. This gives the agent an explicit intermediate representation of the current solution, rather than treating the code as a less clear prompt context. The step-level attention scores further expose which components the ranking model considers most consequential under the current task and workflow context. The largest attention weights identify the workflow components that the ranker considers most performance-sensitive under the current task context, including data augmentation, normalization, and the `efficientnet-b2` backbone. This pattern separates routine implementation components from the parts of the workflow that are likely to affect final performance. In this example, the ranking model identifies the modeling backbone and the surrounding augmentation recipe as the relevant targets for refinement.

In the right panel, we provide a representative edit for the attended modeling step. Given the current `efficientnet-b2` component, TacitFlow retrieves semantically related backbone substitutions from the workflow graph and reranks them under the current workflow context. The reranked edit candidates include replacing `efficientnet-b2` with a pretrained `efficientnet-b3` or `resnet50v2`, which reflects different preferences. Conditioned on the task context and accumulated memory, the proposal agent adopts the `efficientnet-b3` replacement as a minimal, recipe-consistent upgrade, since it builds on the current successful EfficientNet pipeline while increasing capacity and input resolution. Overall, the final edit is supported by a traceable evidence chain: workflow extraction identifies the relevant pipeline steps, attention highlights the performance-sensitive modeling and augmentation components, retrieval surfaces alternative choices, reranking estimates their compatibility with the current workflow, and the proposal agent instantiates the adopted edit with a rationale. This allows the refinement to be attributed to concrete workflow components and alternatives, rather than appearing as an unsupported free-form suggestion. The full demonstration for this competition is provided in Appendix D.

6 Conclusion

We introduced TacitFlow, a framework for turning tacit expert ML knowledge into structured, learned workflow guidance for iterative MLE agents. By constructing a heterogeneous workflow knowledge graph from 131 competitions and 2,681 expert workflows, TacitFlow learns step-level representations and task-conditioned workflow preferences that support a retrieve-edit-rerank loop during solution development. On a diverse 18-competition subset of MLE-Bench, TacitFlow improves automated MLE performance while making each refinement traceable to concrete workflow steps, retrieved edits, and ranking signals. Future work includes extending the framework to additional modalities such as time series and signal data, integrating the learned representations with stronger search frameworks and frontier reasoning models, and scaling the workflow corpus beyond Kaggle.

Acknowledgments

Dongjin Song and Yushan Jiang gratefully acknowledge the support of the National Science Foundation under Grant No. 2338878, as well as the generous research gift from NEC Labs America.

References

[1] Adrien Bardes, Jean Ponce, and Yann LeCun. [n. d.]. VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning. In *International Conference on Learning Representations*.

[2] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. [n. d.]. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. In *The Thirteenth International Conference on Learning Representations*.

[3] Jiefeng Chen, Bhavana Dalvi Mishra, Jaehyun Nam, Rui Meng, Tomas Pfister, and Jinsung Yoon. 2026. MARS: Modular Agent with Reflective Search for Automated AI Research. *arXiv preprint arXiv:2602.02660* (2026).

[4] Shangheng Du, Xiangchao Yan, Dengyang Jiang, Jiakang Yuan, Yusong Hu, Xin Li, Liang He, Bo Zhang, and Lei Bai. 2025. AutoMLGen: Navigating Fine-Grained Optimization for Coding Agents. *arXiv preprint arXiv:2510.08511* (2025).

[5] Haoyang Fang, Boran Han, Nick Erickson, Xiyuan Zhang, Su Zhou, Anirudh Dagar, Jiani Zhang, Ali Caner Turkmen, Cuixiong Hu, Huzefa Rangwala, et al. [n. d.]. MLZero: A Multi-Agent System for End-to-end Machine Learning Automation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

[6] Wei Guan, Jian Cao, Jinyu Cai, Qiqi Cai, Jianqi Gao, and See-Kiong Ng. 2025. GLOW: Graph-Language Co-Reasoning for Agentic Workflow Performance Prediction. *arXiv preprint arXiv:2512.15751* (2025).

[7] 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 *International Conference on Machine Learning*. PMLR, 16813–16848.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[9] Shengran Hu, Cong Lu, and Jeff Clune. [n. d.]. Automated Design of Agentic Systems. In *The Thirteenth International Conference on Learning Representations*.

[10] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation. In *International Conference on Machine Learning*. PMLR, 20271–20309.

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

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] Annan Li, Chufan Wu, Zengle Ge, Yee Hin Chong, Zhinan Hou, Lizhe Cao, Cheng Ju, Jianmin Wu, Huaiming Li, Haobo Zhang, et al. 2025. The fm agent. *arXiv preprint arXiv:2510.26144* (2025).

[14] Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Siheng Chen, et al. 2025. Ml-master: Towards ai-for-ai via integration of exploration and reasoning. *arXiv preprint arXiv:2506.16499* (2025).

[15] Alireza Nadafian, Alireza Mohammadshahi, and Majid Yazdani. 2026. KAPSO: A Knowledge-grounded framework for Autonomous Program Synthesis and Optimization. *arXiv preprint arXiv:2601.21526* (2026).

[16] Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan O Arik, and Tomas Pfister. [n. d.]. MLE-STAR: Machine Learning Engineering Agent via Search and Targeted Refinement. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

[17] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[18] Yixin Ou, Yujie Luo, Jingsheng Zheng, Lanning Wei, Zhuoyun Yu, Shuofei Qiao, Jintian Zhang, Da Zheng, Yuren Mao, Yunjun Gao, et al. 2025. Automind: Adaptive knowledgeable agent for automated data science. *arXiv preprint arXiv:2506.10974* (2025).

[19] Rushi Qiang, Yuchen Zhuang, Yinghao Li, Dingu Sagar VK, Rongzhi Zhang, ChangHao Li, Ian Shu-Hei Wong, Sherry Yang, Percy Liang, Chao Zhang, et al. [n. d.]. MLE-Dojo: Interactive Environments for Empowering LLM Agents in Machine Learning Engineering. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[20] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 3982–3992.

[21] Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, et al. 2025. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. *arXiv preprint arXiv:2507.02554* (2025).

[22] Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. 2026. Multi-View Encoders for Performance Prediction in LLM-Based Agentic Workflows. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=7oeKDZsmWp

[23] Chunhui Wan, Xunan Dai, Zhuo Wang, Minglei Li, Yanpeng Wang, Yinan Mao, Yu Lan, and Zhiwen Xiao. 2025. Loongflow: Directed evolutionary search via a cognitive plan-execute-summarize paradigm. *arXiv preprint arXiv:2512.24077* (2025).

[24] Yue Wang, Hung Le, Akhilesh Gotmare, Nghi Bui, Junnan Li, and Steven Hoi. 2023. Codet5+: Open code large language models for code understanding and generation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*. 1069–1088.

[25] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*. 1192–1199.

[26] Xu Yang, Xiao Yang, Shikai Fang, Yifei Zhang, Jian Wang, Bowen Xian, Qizheng Li, Jingyuan Li, Minrui Xu, Yuante Li, et al. 2025. R&D-Agent: An LLM-Agent Framework Towards Autonomous Data Science. *arXiv preprint arXiv:2505.14738* (2025).

[27] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. [n. d.]. AFlow: Automating Agentic Workflow Generation. In *The Thirteenth International Conference on Learning Representations*.

[28] Yuanshuo Zhang, Yuchen Hou, Bohan Tang, Shuo Chen, Muhan Zhang, Xiaowen Dong, and Siheng Chen. 2025. FLORA: GNNs as Predictors of Agentic Workflow Performances. In *The Fourth Learning on Graphs Conference*. https://openreview.net/forum?id=RmlDI5kUxa

[29] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131* (2020).

[30] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. GPTSwarm: Language Agents as Optimizable Graphs. In *International Conference on Machine Learning*. PMLR, 62743–62767.

A Algorithm

The two-stage offline training and online deployment of TacitFlow are described in Algorithm 1.

B Workflow Collection and Knowledge Graph Construction

To support knowledge-guided machine learning engineering, we build a workflow knowledge graph from expert competition solutions across multiple data modalities. Our objective is to convert raw notebook solutions into reusable structured workflows that preserve both the semantics of individual steps and the dependency patterns among them. Starting from public competition notebooks, we extract workflow traces, canonicalize heterogeneous step descriptions, and merge them into modality-specific directed graphs. The resulting corpus spans image, tabular, and text machine learning tasks and provides a structured substrate for downstream representation learning, workflow ranking, and agentic refinement.

B.1 Workflow Collection

For each target modality, we begin from the public Kaggle competition catalog and issue a curated list of seed keywords describing the modality task families (*e.g.*, classification, detection, and segmentation for image; sentiment, translation, and question-answering for text; regression and recommendation for tabular). Returned competitions are filtered to those whose problem statement matches the target modality, and the union is taken across seeds. To prevent benchmark contamination, we exclude all competitions that appear in MLE-Bench before crawling kernels or constructing ranking labels. Thus, both the workflow graph and the supervision used by the workflow ranking model are built from competitions disjoint from

Algorithm 1 TacitFlow: Knowledge-Grounded Machine Learning Engineering

Initialization: Task t, dev data $\mathcal{D}_{\text{dev}}$, workflow graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, budget B
Input: Best accepted solution $\hat{c}$

▷ Stage 1a: Contrastive node encoder pretraining
Generate augmented views $\tilde{\mathcal{G}}_1, \tilde{\mathcal{G}}_2$ via edge dropping and feature masking
Train encoder f_θ on $(\tilde{\mathcal{G}}_1, \tilde{\mathcal{G}}_2)$ with the contrastive objective in (4.1)
Compute pretrained node embeddings $\{\mathbf{h}_v\}_{v \in \mathcal{V}}$
▷ Stage 1b: Task-conditioned workflow ranking
Form per-competition workflow rankings $\{\sigma_c\}_{c \in C}$ from leaderboards
Train ranker s_ϕ on $\{\sigma_c\}$ via the ListMLE objective in (4.2)
▷ Stage 2: Online knowledge-grounded refinement
Initialize baseline $\hat{c} \leftarrow c_0$, $\hat{m} \leftarrow -\infty$, history $\mathcal{H} \leftarrow \emptyset$, edit memory $\mathcal{L} \leftarrow \emptyset$
while budget B not exhausted **do**
  Extract structured workflow $w \leftarrow \text{Extract}(\hat{c})$
  for each step $v \in \mathcal{V}_w$ **do**
    Map v to embedding space via exact / fuzzy / constructed matching
    $\mathcal{N}_{\text{ref}}(v) \leftarrow \{u \in \mathcal{V} : \text{sim}(\mathbf{h}_v, \mathbf{h}_u) \geq \delta,\ u \neq v\}$
  end for
  Construct candidates $\widetilde{\mathcal{W}}$ by applying each retrieved edit to w
  $\mathcal{W}^\star \leftarrow \bigcup_c \text{top-}k\{s_\phi(\tilde{w}, t) : \text{cat}(\tilde{w}) = c\}$
  $\gamma \leftarrow \text{Reason}(\mathcal{W}^\star, \mathcal{L}, t)$ ▷ Reason over edits using edit memory
  $i \leftarrow \text{Propose}(\kappa, \mathcal{H})$, where $\kappa \triangleq (\mathcal{W}^\star, \gamma)$ ▷ Generate, score, select/ensemble
  $c \leftarrow \text{Implement}(i; \hat{c})$
  $m \leftarrow \text{Evaluate}(c, \mathcal{D}_{\text{dev}})$
  if $m > \hat{m}$ **then**
    $\hat{c} \leftarrow c,\ \hat{m} \leftarrow m$
  end if
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(i, c, m)\}$
  $\ell \leftarrow \text{Summarize}(\widetilde{\mathcal{W}}, i, \Delta c, m)$ ▷ Proposal, code change, validation outcome
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{\ell\}$
end while
return $\hat{c}$

the evaluation tasks. This ensures that TacitFlow learns transferable workflow preferences rather than task-specific solutions or leaderboard information from the benchmark competitions.

For every retained competition, we crawl the public kernels, tag each one as a *solution*, *scored*, or *unscored* kernel from public metadata, and concatenate the kernels in that order into a per-modality link list. The resulting position of a workflow within its own source competition acts as a relative performance label for the downstream ranking model.

We collect workflows from public competition solutions across three modalities: image, tabular, and text. In total, the corpus contains 131 competitions and 2,681 workflows, comprising 883 image workflows from 48 competitions, 1,226 tabular workflows from 55 competitions, and 572 text workflows from 28 competitions. The competitions are disjoint across modalities, which allows each graph to capture modality-specific workflow regularities without overlap in source tasks. Across all modalities, the collected workflows exhibit substantial scale and structural diversity. In total, the corpus contains 10,126 workflow steps and 32,920 directed edges. The average workflow length is relatively consistent across modalities, ranging from 18.3 to 20.4 steps, while the average number of workflows per competition ranges from 18.4 to 22.3.

The workflow corpus covers a diverse range of machine learning settings, including classification, regression, detection, segmentation, retrieval, generation, forecasting, clustering, and ranking. It also spans multiple application domains such as healthcare, earth science, automotive, commerce, finance, education, and information retrieval. This diversity is important for learning reusable workflow components that generalize across tasks while preserving modality-specific structure. The detailed list is provided in Table 6.

B.2 Workflow Extraction and Canonicalization

Notebook solutions often contain noisy execution traces, helper cells, debugging code, and repeated exploratory steps. To obtain reusable workflow units, we canonicalize each workflow into a sequence of semantically meaningful pipeline steps. Each step is represented as a structured node with five attributes: category: coarse pipeline stage, technique: main operation or method name, code: associated code snippet, notes: natural language description. To standardize workflows across notebooks, we use six coarse stage categories: Data Read, Data Preparing, Modeling, Training, Evaluation, and Analysis. This shared vocabulary reduces superficial variation across notebooks while preserving the functional role.

Table 6: Kaggle competitions used for workflow collection.

Modality	Competition IDs
Image (48)	airbus-ship-detection; benetech-making-graphs-accessible; carvana-image-masking-challenge; cifar-10; cvpr-2018-autonomous-driving; data-science-bowl-2017; data-science-bowl-2018; diabetic-retinopathy-classification-2; diabetic-retinopathy-detection; digit-recognizer; dstl-satellite-imagery-feature-detection; generative-dog-images; global-wheat-detection; google-ai-open-images-object-detection-track; google-ai-open-images-visual-relationship-track; google-universal-image-embedding; hpa-single-cell-image-classification; human-protein-atlas-image-classification; image-matching-challenge-2022; imaterialist-challenge-fashion-2018; imaterialist-challenge-furniture-2018; imet-2019-fgvc6; inclusive-images-challenge; isic-2024-challenge; jaguar-re-id; landmark-recognition-2019; landmark-recognition-challenge; landmark-retrieval-2020; landmark-retrieval-2021; landmark-retrieval-challenge; mayo-clinic-strip-ai; open-images-2019-object-detection; open-images-2019-visual-relationship; petfinder-adoption-prediction; pku-autonomous-driving; planet-understanding-the-amazon-from-space; planttraits2024; recodai-luc-scientific-image-forgery-detection; recursion-cellular-image-classification; rsna-intracranial-aneurysm-detection; sartorius-cell-instance-segmentation; sp-society-camera-model-identification; state-farm-distracted-driver-detection; tensorflow-great-barrier-reef; tpu-getting-started; ultrasound-nerve-segmentation; understanding_cloud_organization; vesuvius-challenge-surface-detection
Text (28)	asap-aes; avito-duplicate-ads-detection; classification-of-math-problems-by-kasut-academy; coleridgeinitiative-show-us-the-data; commonlit-evaluate-student-summaries; commonlitreadabilityprize; crowdflower-search-relevance; deep-past-initiative-machine-translation; eedi-mining-misconceptions-in-mathematics; fake-or-real-the-impostor-hunt; feedback-prize-2021; feedback-prize-effectiveness; feedback-prize-english-language-learning; gendered-pronoun-resolution; jigsaw-toxic-severity-rating; llm-agentic-legal-information-retrieval; llm-classification-finetuning; llm-detect-ai-generated-text; llm-prompt-recovery; llms-you-cant-please-them-all; map-charting-student-math-misunderstandings; nbme-score-clinical-patient-notes; nlp-getting-started; quora-question-pairs; sentiment-analysis-on-movie-reviews; super-ai-ss-5-named-entity-recognition; tradeshift-text-classification; uspto-explainable-ai
Tabular (55)	car-becho-paisa-paao; cat-in-the-dat; cat-in-the-dat-ii; equity-post-HCT-survival-predictions; home-depot-product-search-relevance; house-prices-advanced-regression-techniques; hull-tactical-market-prediction; instacart-market-basket-analysis; linking-writing-processes-to-writing-quality; mlsp-2014-mri; optiver-realized-volatility-prediction; otto-group-product-classification-challenge; otto-recommender-system; playground-series-s3e1; playground-series-s3e11; playground-series-s3e12; playground-series-s3e13; playground-series-s3e17; playground-series-s3e21; playground-series-s3e26; playground-series-s3e3; playground-series-s3e5; playground-series-s3e6; playground-series-s3e7; playground-series-s3e8; playground-series-s4e2; playground-series-s4e3; playground-series-s4e6; playground-series-s5e11; playground-series-s5e2; playground-series-s5e3; playground-series-s5e4; playground-series-s5e5; playground-series-s5e6; playground-series-s5e7; playground-series-s5e8; playground-series-s6e1; stock-pledge-financing-default-prediction-2026; tabular-playground-series-apr-2022; tabular-playground-series-aug-2021; tabular-playground-series-aug-2022; tabular-playground-series-feb-2021; tabular-playground-series-feb-2022; tabular-playground-series-jan-2021; tabular-playground-series-jan-2022; tabular-playground-series-jul-2022; tabular-playground-series-jun-2021; tabular-playground-series-jun-2022; tabular-playground-series-mar-2022; tabular-playground-series-may-2021; tabular-playground-series-nov-2022; tabular-playground-series-oct-2022; tabular-playground-series-sep-2021; tabular-playground-series-sep-2022; titanic

Extraction proceeds in two passes. In the first pass, each kernel is parsed into an ordered sequence of code cells, and a single LLM call converts these cells into the structured workflow described above. The prompt asks the LLM to describe the actual operation performed by each cell. If a cell contains multiple distinct operations, it is split into multiple workflow nodes. In the second pass, we canonicalize technique names across notebooks. Variants that refer to the same method are grouped into a single canonical key using a lightweight LLM-based canonicalization step. We save these canonicalization results and reuse them in later runs, so the same technique variants are mapped consistently. The resulting nodes are also well populated: most nodes contain a code snippet, a natural language note, and a canonical technique name. Thus, each node combines categorical, textual, and code-based information, making it naturally multi-modal.

B.3 Knowledge Graph Construction

Given the canonicalized workflows, we construct a directed workflow knowledge graph for each modality. For a modality-specific graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, each node $v \in \mathcal{V}$ corresponds to a canonical workflow step, and each directed edge $(u, v) \in \mathcal{E}$ indicates that step v follows step u in at least one workflow. Thus, edges encode stepwise dependencies and execution order as observed in expert solutions. A workflow w is represented as a directed subgraph $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w) \subseteq \mathcal{G}$, where $\mathcal{V}_w$ is the set of workflow steps and $\mathcal{E}_w$ contains the observed transitions among them. Beyond intra-workflow transitions, we align steps that implement the same canonical technique across notebooks and merge them into shared graph nodes. The resulting graph aggregates cross-workflow dependencies and reveals reusable pipeline patterns beyond individual notebooks. The structural composition of the graph reveals that preprocessing dominates all three modalities, followed by modeling, evaluation, and training. This is consistent with practical competition pipelines, where data preparation often accounts for a large fraction of solution effort.

B.4 Graph Cleaning and Quality Control

Notebook-derived workflows over-produce nodes: surface-form drift in technique names, near-synonyms, and competition-specific plumbing all leave nodes that are technically distinct but functionally identical or non-transferable. The code field, which is consumed by a mean-pooled code encoder for retrieval, is also sensitive to boilerplate noise. We therefore apply a sequence of cleaning passes before finalizing the corpus.

Embedding-based deduplication. We compute sentence embeddings of the concatenated technique name and notes for each node,

Table 7: Step category distribution across modalities. Values are counts with percentages in parentheses.

Category	Image	Tabular	Text
Data Preparing	1,635 (46.2%)	1,950 (50.8%)	1,240 (45.0%)
Modeling	936 (26.5%)	781 (20.4%)	785 (28.5%)
Evaluation	565 (16.0%)	549 (14.3%)	361 (13.1%)
Training	388 (11.0%)	488 (12.7%)	353 (12.8%)
Analysis	11 (0.3%)	67 (1.7%)	14 (0.5%)

and merge node pairs whose cosine similarity exceeds a conservative threshold. This pass removes near-duplicates that survive canonicalization, *e.g.*, minor lexical variants of learning-rate schedules or splitting strategies.

Singleton classification. Nodes supported by only a single notebook are the dominant source of non-transferable noise. We classify each such node with multiple LLM votes under complementary prompt variants and apply a consensus rule, with disagreements surfaced for manual review. The classifier labels each node as keep (a generic technique likely to recur, *e.g.*, a standard augmentation or loss), drop (competition-specific plumbing, infrastructure code, or a verbose variant already covered by an existing multi-use node), or merge (folded into a named target node). For the image modality, deduplication and singleton filtering together reduce the graph from 5,475 raw nodes to 3,536 deployed nodes, while preserving all 48 competitions and all 883 workflows; analogous reductions are obtained for text and tabular.

Code attachment and quality control. After constructing the canonical workflow graph, we attach a representative code snippet to each surviving node by revisiting its source notebooks. Each occurrence of a node contributes one candidate snippet. For nodes with multiple occurrences, an LLM selects the candidate that best represents the node's canonical technique and functional role. As the retrieval module embeds the code field of each node, the attached snippet should contain technique-specific signal. To ensure this, we replace full-node rewriting with a lightweight three-stage quality-control pipeline. In Select, for nodes with multiple source occurrences, an LLM selects the candidate snippet that best represents the node's canonical technique and functional role, favoring concise snippets that are already informative. In Generate, for nodes whose observed candidates are missing, empty, or trivially uninformative, an LLM generates a short Python snippet that directly expresses the intended operation. The prompt discourages imports, wrappers, and unrelated setup code. In Review, for shared nodes, an LLM makes a binary keep/rewrite decision. The review prompt favors keeping the original snippet when it contains sufficient technique signal and rewrites only when the snippet is too noisy or uninformative.

Overall, our construction process is designed to preserve three properties that are critical for downstream agent use: **(i)** *semantic richness*, by retaining technique names, code snippets, and natural language notes at the node level; **(ii)** *structural faithfulness*, by encoding execution order as directed dependencies; and **(iii)** *reusability*, by canonicalizing step names and merging repeated solution patterns across workflows. The resulting knowledge graph provides a compact and structured view of expert machine learning workflows. Its nodes capture reusable building blocks such as preprocessing operations, model classes, training strategies, and evaluation routines, while its edges capture how these blocks are composed into valid pipelines. Full workflow traces are also preserved, allowing solutions to be retrieved, ranked, and adapted for new tasks. This structure is especially useful for machine learning engineering agents. Instead of exploring the solution space from scratch, an agent can reason over existing workflow components and their relations, retrieve promising pipeline structures, and modify them transparently. In this sense, the workflow graph serves not only as a memory of past expert solutions, but also as a structured search space for knowledge-guided refinement.

Table 8: Mapping between shortened competition names used in the main text and official MLE-Bench competition IDs.

Short name	MLE-Bench competition ID
dog-breed	dog-breed-identification
plant-pathology-20	plant-pathology-2020-fgvc7
dogs-vs-cats	dogs-vs-cats-redux-kernels-edition
tgs-salt	tgs-salt-identification-challenge
uw-gi-tract	uw-madison-gi-tract-image-segmentation
kuzushiji	kuzushiji-recognition
denoising-dirty	denoising-dirty-documents
spooky-author	spooky-author-identification
detect-insults	detecting-insults-in-social-commentary
jigsaw-toxic	jigsaw-toxic-comment-classification-challenge
us-patent	us-patent-phrase-to-phrase-matching
essay-score-2	learning-agency-lab-automated-essay-scoring-2
stanford-vaccine	stanford-covid-vaccine
text-norm-russian	text-normalization-challenge-russian-language
random-acts	random-acts-of-pizza
tps-dec-2021	tabular-playground-series-dec-2021
nomad2018	nomad2018-predict-transparent-conductors
tps-may-2022	tabular-playground-series-may-2022

B.5 Ablation of Offline Training Components

In this subsection, we provide additional ablation studies regarding different components as well as feature modalities to assess the downstream ranking performance. Table 12 reports the full results across five metrics for in-corpus data and Table 13 reports those for out-of-corpus. We also provide other ablations including the feature importance (technique name, code, notes) across three image, text, and tabular modalities, in Table 9, 10, 11 of Appendix B.5.

C Prompts

We first provide the key components of prompts for online MLE tasks, including edit reasoning in Figure 5, proposal agent with edit guidance in Figure 6, and the coding agent in Figure 7. Second, we provide the necessary prompts for workflow extraction from Kaggle solutions and workflow knowledge graph building in Figure 8 and Figure 9. Lastly, we provide the prompts for code feature cleansing in Figure 10.

D Full Demonstration

Please refer to this link for the full demonstration on *plant-pathology-2020-fgvc7* : https://anonymous.4open.science/r/TacitFlow-2A19/README.md

Table 9: Image modality: ablation of node-level input features (in-corpus, 5-fold CV × 3 seeds).

Features				Metrics				
Cat.	Tech.	Notes	Code	NDCG@5	NDCG@10	MRR	Top-1	Top-3
✓	✓	✗	✗	0.872 ± 0.032	0.884 ± 0.024	0.440 ± 0.138	0.236 ± 0.162	0.532 ± 0.176
✓	✗	✓	✗	0.874 ± 0.035	0.889 ± 0.025	0.435 ± 0.127	0.230 ± 0.163	0.530 ± 0.160
✓	✗	✗	✓	0.878 ± 0.035	0.888 ± 0.029	0.474 ± 0.127	0.272 ± 0.152	0.599 ± 0.157
✓	✓	✓	✗	0.882 ± 0.041	0.896 ± 0.027	0.469 ± 0.129	0.259 ± 0.149	0.599 ± 0.182
✓	✓	✗	✓	0.884 ± 0.033	0.891 ± 0.024	0.472 ± 0.128	0.238 ± 0.181	**0.664 ± 0.145**
✓	✗	✓	✓	0.885 ± 0.029	0.894 ± 0.022	0.451 ± 0.124	0.246 ± 0.159	0.555 ± 0.170
✓	✓	✓	✓	**0.897 ± 0.037**	**0.904 ± 0.022**	**0.488 ± 0.132**	**0.300 ± 0.150**	0.612 ± 0.200

Table 10: Text modality: ablation of node-level input features (in-corpus, 5-fold CV × 3 seeds).

Features				Metrics				
Cat.	Tech.	Notes	Code	NDCG@5	NDCG@10	MRR	Top-1	Top-3
✓	✓	✗	✗	0.791 ± 0.066	0.777 ± 0.047	0.385 ± 0.202	0.185 ± 0.244	0.378 ± 0.190
✓	✗	✓	✗	0.793 ± 0.061	0.780 ± 0.045	0.359 ± 0.121	0.178 ± 0.155	0.358 ± 0.143
✓	✗	✗	✓	0.797 ± 0.061	0.778 ± 0.052	0.392 ± 0.165	0.193 ± 0.200	0.453 ± 0.233
✓	✓	✓	✗	0.804 ± 0.067	0.778 ± 0.057	0.407 ± 0.174	0.213 ± 0.205	0.469 ± 0.232
✓	✓	✗	✓	0.791 ± 0.068	0.783 ± 0.051	0.360 ± 0.099	0.176 ± 0.106	0.391 ± 0.132
✓	✗	✓	✓	0.783 ± 0.066	0.774 ± 0.046	0.394 ± 0.095	0.200 ± 0.133	0.473 ± 0.135
✓	✓	✓	✓	**0.833 ± 0.051**	**0.813 ± 0.039**	**0.434 ± 0.139**	**0.216 ± 0.165**	**0.529 ± 0.258**

Table 11: Tabular modality: ablation of node-level input features (in-corpus, 5-fold CV × 3 seeds).

Features				Metrics				
Cat.	Tech.	Notes	Code	NDCG@5	NDCG@10	MRR	Top-1	Top-3
✓	✓	✗	✗	0.768 ± 0.066	0.764 ± 0.051	0.466 ± 0.115	0.279 ± 0.112	0.588 ± 0.193
✓	✗	✓	✗	0.752 ± 0.100	0.752 ± 0.071	0.443 ± 0.184	0.236 ± 0.159	0.594 ± 0.318
✓	✗	✗	✓	0.801 ± 0.043	0.785 ± 0.036	0.517 ± 0.102	0.333 ± 0.127	0.636 ± 0.156
✓	✓	✓	✗	0.782 ± 0.080	0.773 ± 0.051	0.494 ± 0.164	0.303 ± 0.171	0.642 ± 0.254
✓	✓	✗	✓	0.815 ± 0.054	0.784 ± 0.038	**0.543 ± 0.090**	0.333 ± 0.108	0.703 ± 0.157
✓	✗	✓	✓	0.784 ± 0.081	0.773 ± 0.054	0.459 ± 0.140	0.242 ± 0.131	0.642 ± 0.231
✓	✓	✓	✓	**0.814 ± 0.058**	**0.790 ± 0.044**	0.540 ± 0.109	**0.342 ± 0.113**	**0.720 ± 0.215**

Table 12: Model ablation study of TacitFlow (in-corpus, 5-fold CV × 3 seeds).

Modality	Condition	NDCG@5	NDCG@10	MRR	Top-1	Top-3
Image	w/o Pretraining	0.888 ± 0.032	0.895 ± 0.021	0.449 ± 0.133	0.228 ± 0.149	0.584 ± 0.195
	w/o VICReg	0.880 ± 0.026	0.889 ± 0.018	0.450 ± 0.118	0.242 ± 0.155	0.574 ± 0.128
	w/o Attention	0.889 ± 0.022	0.894 ± 0.020	0.459 ± 0.122	0.241 ± 0.153	0.575 ± 0.159
	Ours	**0.897 ± 0.037**	**0.904 ± 0.022**	**0.488 ± 0.132**	**0.300 ± 0.150**	**0.612 ± 0.180**
Text	w/o Pretraining	0.786 ± 0.061	0.776 ± 0.051	0.315 ± 0.110	0.125 ± 0.135	0.324 ± 0.181
	w/o VICReg	0.804 ± 0.067	0.782 ± 0.054	0.427 ± 0.073	0.203 ± 0.141	0.522 ± 0.135
	w/o Attention	0.826 ± 0.060	0.800 ± 0.046	0.407 ± 0.104	0.189 ± 0.142	**0.547 ± 0.181**
	Ours	**0.833 ± 0.051**	**0.813 ± 0.039**	**0.434 ± 0.139**	**0.216 ± 0.165**	0.529 ± 0.258
Tabular	w/o Pretraining	0.775 ± 0.093	0.762 ± 0.059	0.493 ± 0.150	0.280 ± 0.141	0.673 ± 0.231
	w/o VICReg	0.795 ± 0.055	0.777 ± 0.040	0.534 ± 0.113	0.333 ± 0.143	0.709 ± 0.204
	w/o Attention	0.769 ± 0.079	0.763 ± 0.050	0.475 ± 0.166	0.273 ± 0.159	0.636 ± 0.220
	Ours	**0.814 ± 0.058**	**0.790 ± 0.044**	**0.540 ± 0.109**	**0.342 ± 0.113**	**0.720 ± 0.215**

Table 13: Model ablation study of TacitFlow (OOD Evaluation on MLE-Bench runs, 3 seeds).

Modality	Condition	NDCG@5	NDCG@10	MRR	Top-1	Top-3
Image	w/o Pretraining	0.645 ± 0.019	0.721 ± 0.006	0.258 ± 0.012	0.060 ± 0.045	0.202 ± 0.110
	w/o VICReg	0.650 ± 0.002	0.736 ± 0.007	0.256 ± 0.008	0.073 ± 0.015	0.260 ± 0.015
	w/o Attention	0.675 ± 0.022	**0.763 ± 0.006**	0.242 ± 0.008	0.073 ± 0.015	0.271 ± 0.059
	Ours	**0.688 ± 0.016**	**0.763 ± 0.016**	**0.284 ± 0.025**	**0.183 ± 0.047**	**0.344 ± 0.026**
Text	w/o Pretraining	0.767 ± 0.021	0.825 ± 0.009	0.315 ± 0.025	0.079 ± 0.022	0.508 ± 0.011
	w/o VICReg	0.746 ± 0.025	0.791 ± 0.014	0.391 ± 0.020	0.230 ± 0.030	0.413 ± 0.040
	w/o Attention	0.724 ± 0.017	0.794 ± 0.008	0.389 ± 0.028	0.246 ± 0.049	0.421 ± 0.045
	Ours	**0.783 ± 0.023**	**0.829 ± 0.019**	**0.519 ± 0.008**	**0.381 ± 0.019**	**0.587 ± 0.092**
Tabular	w/o Pretraining	0.561 ± 0.013	0.703 ± 0.011	0.178 ± 0.020	0.000 ± 0.000	0.083 ± 0.059
	w/o VICReg	0.567 ± 0.042	0.720 ± 0.020	0.194 ± 0.012	0.000 ± 0.000	0.208 ± 0.059
	w/o Attention	0.638 ± 0.019	0.788 ± 0.018	0.281 ± 0.027	0.083 ± 0.059	**0.375 ± 0.000**
	Ours	**0.650 ± 0.024**	**0.791 ± 0.027**	**0.433 ± 0.023**	**0.333 ± 0.059**	**0.375 ± 0.000**

Key Prompt of Edit Reasoning

****System.****
You are an experienced ML research advisor reviewing a candidate-edit menu for a Kaggle [MODALITY] pipeline. The menu was retrieved by an external workflow ranker from a knowledge base of past Kaggle solutions, then will be passed to a downstream hypothesis generator.

Your role: ADVISORY, not prescriptive. The downstream generator decides freely. Your job is to enrich the menu with one short advisory note per option so the generator can engage more thoughtfully.

Allowed actions per option:
(a) Caveat — flag if the option resembles a recently failed attempt or was rejected in an earlier loop of this run.
(b) Pair — when two options would naturally combine given the base workflow, mention this briefly.
(c) Frame — provide context on when this technique typically helps for this scenario (task type, metric, data size, time budget).

Forbidden:
- Do NOT tell the generator to adopt or reject a specific option.
- Do NOT rewrite option content, rescore, or rerank.
- Do NOT invent options outside the menu.
- Do NOT generate a hypothesis.

Output: reproduce the menu exactly (preserve category headers and option numbering). Under each option add one short advisory paragraph (2-3 sentences). Omit the paragraph if you have nothing useful to say.
Keep total output under 800 words.

****User.****
 # Scenario Summary
[SCENARIO_SUMMARY]

 # Base Workflow
[BASE_WORKFLOW]

 # Last Loop Outcome
[LAST_OUTCOME]

 # Recent Lessons from this Run (most recent first)
[MEMORY_RECENT]

 # Candidate Menu (ranker output, ranked by category)
[CANDIDATES]

Reproduce the menu and add a short Advisory paragraph below each option where useful. Stay strictly advisory.

Figure 5: Prompt used by the proposal agent to reason over ranked workflow edits.

Key Prompt of Proposal Agent with Edit Guidance

****System.****
You are a Kaggle Grandmaster and expert ML engineer with deep expertise in statistics, machine learning, and competition optimization.
The user is iteratively improving a Kaggle competition implementation. Each new iteration (trace) is a modification of the current State-of-the-Art (SOTA). If a new trace surpasses the current SOTA, it becomes the new SOTA. Otherwise, it's a failed experiment.

You will be provided with:
1. A detailed competition description and identified challenges from history.
2. A history of previous successful and failed experiments and their feedback.
3. The current SOTA implementation and feedback.
4. A set of Ranked Candidate Edits derived from a knowledge base of Kaggle workflows. Each edit is a technique-level replacement in the current workflow, scored by a trained workflow quality model.

Task 1 - Hypothesis Proposal: for each Identified Challenge, propose one specific, testable hypothesis.

Task 2 - Hypothesis Evaluation: assign each hypothesis a component tag and score it on five 1-10 dimensions (alignment, expected impact, novelty, feasibility, risk-reward balance).

Workflow Improvement Suggestions from Knowledge Base.
You are also provided with ranked candidate edits derived from successful Kaggle workflows. These suggest technique replacements at specific workflow positions, scored by a trained workflow quality model. If a suggested edit is compatible with the current competition's data type, model framework, and pipeline constraints, consider adopting it as the basis for one of your hypotheses.

****User.****
 # Scenario Description
[SCENARIO_DESC]

 # Previous Experiments and Feedbacks
[EXP_AND_FEEDBACK_LIST]

 # Current SOTA Implementation
[SOTA_EXP_DESC]

 # Identified Challenges
[PROBLEMS]

 # Ranked Candidate Edits
The section below shows: (1) the current workflow baseline, and (2) suggested edits ranked by an external workflow ranker trained on successful Kaggle workflows.

Instructions:
- Review the current workflow steps to understand the baseline.
- Choose at most ONE edit to implement as your core hypothesis change.
- The edit suggestions are based on what worked well in similar competitions.
- If you choose none, explicitly state why they are unsuitable.

[RANKED_EDIT_SUGGESTIONS]

At runtime, [RANKED_EDIT_SUGGESTIONS] renders as the current numbered workflow plus retrieved edits grouped by category, each formatted as "Option N: replace [OLD] → [NEW] at step [TARGET_IDX]".

Figure 6: Prompt used by the proposal agent to generate candidate hypotheses based on edit guidance.

Key Prompt of Coding Agent

****System.****
You are a grandmaster-level data scientist and machine learning engineer with deep expertise in statistics, mathematics, and computer science.
Your task is to generate robust, debuggable, iteration-friendly code for data science pipelines, following a strict, stepwise process.
You are working on sample datasets and your code will go through automated iterations. Design your code to be iteration-friendly with comprehensive print statements and clear debugging information.

 # Task Description
[TASK_DESC]

Runtime Environment
[RUNTIME_ENVIRONMENT]

Package Information
[PACKAGE_INFO]

Hyperparameters Specification
Follow hyperparameter choices specified in the task unless they are unreasonable or incorrect.
 # Specification your code should follow
[SPEC]

 # Workflow Overview
Data Loading
- Load the dataset strictly from the configured input path. Do NOT load from the current directory.
- Beyond data loading, avoid try-except blocks to hide or suppress errors. All errors should be properly diagnosed and fixed at source.

Exploratory Data Analysis
Output a plain-text EDA section to stdout within the markers:
=== Start of EDA part ===
[EDA content: shape, dtypes, missing values, target distribution, feature analysis]
=== End of EDA part ===

Debug Mode
The code is invoked as `python main.py --debug`. In debug mode, sample 10% of training data and run minimum epochs, then print:
=== Start of Debug Information ===
debug_time: <seconds>
estimated_time: <seconds for full run>
=== End of Debug Information ===
In full mode, use all data and full epochs.

General Guidelines
1. Code correctness is the top priority. Ensure your code is runnable and produces the expected output even if some task requirements are not fully met.
2. Use print() for all output; do not use the logging module.
3. Avoid all hard-coded values (e.g., fixed dataset sizes). Always use proportions for data splitting, never absolute numbers.
4. Add informative print statements at key steps.
5. For model training, use reasonable epoch numbers and implement early stopping. Save best model checkpoints based on validation performance.
6. Except in debug mode, ALWAYS use all available data; do not sample or subset due to resource limitations.
7. Do not use tqdm or progress bars.
8. Try-except blocks are ONLY allowed when reading files. Assert statements are PROHIBITED throughout.
9. ALWAYS use the best saved model for predictions. NEVER create dummy submissions (all 1s, random values). If training fails, report failure honestly rather than generating fake submission files.
10. Generate the complete code, not partial code.
11. If the task contains user instructions, follow them - they have the highest priority.

****User.****
 # Competition Information
[COMPETITION_INFO]

 # Data Folder Description
[FOLDER_SPEC]

 # Former code (if iterating)
[LATEST_CODE]

Feedback to former code
[LATEST_CODE_FEEDBACK]

Before modifying the code, analyze the feedback and identify no more than three key areas requiring changes. Prioritize critical issues affecting execution, correctness, or stability; preserve existing working components.

Figure 7: Prompt used by the coding agent to generate robust, debuggable, and iterative MLE code.

Key Prompts of Workflow Extraction and Workflow Knowledge Graph Building (Part 1)

****A.4 Notebook Triage.****
You judge if the document (notebook code, GitHub README, or Kaggle solution writeup) provides enough information to understand and reproduce ONE coherent final workflow that yields the official result.

Rules
1. Suitable if the document clearly shows how data, modeling, and prediction connect to produce the final result.
2. Ensembles, stacks, or cascades that merge into one final output count as ONE workflow.
3. Not suitable if multiple unrelated final pipelines are presented without a single chosen one.
4. Pure analysis or visualization without modeling/prediction -> not suitable.
5. Never infer missing steps; rely only on explicitly provided content.

Input
[NOTEBOOK_CLUES]

Output
decision: "suitable" | "not_suitable", reason: "<=120 chars>"

****A.5 Workflow Extraction (PLAN pass).****
You are extracting a canonical workflow plan from a Kaggle source document.

[INSIGHTS] curated claims and techniques mined from text/discussion.

Derive an ordered list of pipeline steps covering the full workflow implied by [INSIGHTS]. Each step is a (category, technique) pair.

Categories (assign exactly one per step)
- Data Read | Data Preparing | Modeling | Training | Evaluation

Coverage. Every workflow MUST include at least one step for each of the following slots, specialized strictly from [INSIGHTS]:
(1) data source / datamix construction
(2) data transformation / augmentation / normalization
(3) feature representation
(4) base modeling
(5) ensemble or meta modeling (if implied)
(6) training scheme
(7) evaluation protocol

Hard constraint: if [INSIGHTS] mention 3+ distinct techniques, output at least 8 pipeline steps; if 6+ items, at least 10.

Rules
- Promote any operation causing a semantic state change in the pipeline into its own step.
- Technique names: unique, concise, lowercased, canonical.
- Exclude generic placeholders unless a specific algorithm is named.
- Maintain a realistic runtime ordering.

Figure 8: Prompts for workflow extraction and workflow knowledge graph construction: notebook triage, step extraction.

Key Prompts of Workflow Extraction and Workflow Knowledge Graph Building (Part 2)

****A.6.1 Cluster naming.****
You standardize step names for a cluster of similar steps.

Goal: produce canonical (category, techniques) pairs that are precise, reproducible, and library-aligned.
- If all excerpts describe one operation -> output ONE canonical item.
- If they describe distinct operations -> output MULTIPLE items in logical runtime order.
- Prefer existing canonical names. Reuse an existing name if it refers to the same technique even if naming or arguments differ. Create a new name only when no existing item captures the same library class or algorithmic intent.

Naming: Category in {Data Read, Data Preparing, Modeling, Training, Evaluation, Analysis}. Technique = library class/method name or canonical algorithm name (sklearn.preprocessing.StandardScaler.transform, torch.optim.Adam, resnet, lightgbm). One technique per item; expand chained calls into atomic steps.

Target cluster
[CLUSTER_EXCERPTS]

Existing relevant names
[CANDIDATE_EXISTING_NAMES]

****A.6.2 Action selection.****
You decide how to handle ONE target step.
- CATEGORY: [CATEGORY]
- TECHNIQUES: [TECHNIQUES]
- NOTE: [NOTES]
- HYPERPARAM: [HYPS]

Classify into one of:
- "drop" : only structural / I/O / wrapping; no semantic change to data, model, or evaluation.
- "rename_or_split" : ambiguous, generic, user-defined, dataset-specific, or combines multiple operations.
- "proper" : one clear, reproducible, technically meaningful process.

When uncertain, prefer "rename_or_split" over "proper".

****A.6.3 Rename/split refinement.****
You refine ONE target step whose (category, techniques) may need rename and/or split.

Input
{"category":"[CATEGORY]", "techniques":"[TECHNIQUES]", "notes":"[NOTES]", "hyperparams":"[HYPS]", "code":"[CODE_EXAMPLE]"}

Rules
- SPLIT when evidence indicates multiple distinct operations.
- RENAME when the technique is generic or user-defined / dataset-specific.
- KEEP as a single item when already specific and reproducible.
- EXCLUDE utility/housekeeping steps with no effect on data, features, model training, or evaluation.

****A.6.4 Match to global vocabulary.****
You canonicalize ONE operation by matching a TARGET PAIR against existing canonical names, using contextual evidence.

Target pairs (candidates for THIS operation)
[CANDIDATE_PAIRS]

Context
{"notes":"[NOTES]","hyperparams":"[HYPS]","code":"[CODE_EXAMPLE]"}

Existing relevant names (global vocabulary so far)
[EXISTING_NAMES]

Decide
1) Which target pair best describes the operation.
2) Then:
a) REUSE an existing canonical name if it represents the same op.
b) KEEP the chosen pair (possibly with minor normalization).
c) CREATE a new canonical pair only if neither applies.
3) Constraint: at most one "Data Read" step per workflow.

****A.7 Dataset-Task Similarity Scoring.****
You are a meticulous dataset-task similarity judge.

Facets (each scored 0..5, weights in parentheses)
F1. Modality & Channels (30%)
F2. Unit of Analysis & Domain (20%)
F3. Supervision / Target Structure (20%)
F4. Intrinsic Structure (spatial/temporal/hierarchy) (15%)
F5. Distribution & Data Quality (10%)
F6. Metric & Evaluation Protocol (5%)

Per candidate:
1) Assign facet scores f1..f6 in {0..5}.
2) S = 100*(0.30*f1/5 + 0.20*f2/5 + 0.20*f3/5 + 0.15*f4/5 + 0.10*f5/5 + 0.05*f6/5)
3) raw_score = round(S).

Figure 9: Prompts for workflow extraction and workflow knowledge graph construction: canonicalization and step-level similarity scoring.

Key Prompts of Code Quality Control

A.8 Code-Snippet Selection
When a knowledge-graph node aggregates multiple code snippets harvested from different notebooks, picks the single best one.

System.

You are a knowledge graph curator. Select the single best code snippet to represent a technique node. The snippet will be embedded by CodeT5+ (mean pooling over 512 tokens) for similarity retrieval.

A good snippet:
- Is complete, runnable Python (not a fragment or config value).
- Is generic and reusable (no hardcoded competition paths or column names).
- Clearly illustrates the described technique.
- Starts with the most distinctive code (not imports).

Avoid:
- Very short config lines (SEED=42, n_folds=5).
- Entirely commented out code.
- Shell or magic commands.
- Code hardcoded to one specific competition.

User.

Select the single best code snippet for the following node.

Technique: [TECHNIQUE NAME]
Category: [CATEGORY]
Description: [TECHNIQUE DESCRIPTION]

Candidate snippets:

--- Snippet 1 ---
[CANDIDATE SNIPPET 1]

--- Snippet 2 ---
[CANDIDATE SNIPPET 2]
...

Reply with ONLY the snippet number (e.g. "3"). No explanation.

A.9 Code-Snippet Generation
When every candidate snippet for a node is unusable, synthesizes a canonical implementation from the node's metadata.

System.
You are a Python code generator for a machine learning knowledge graph. Given a technique description, write a short, representative Python snippet that will be embedded by CodeT5+ for retrieval.

Rules:
- Valid, runnable Python (not pseudocode).
- Generic variable names (X, y, df, train, test, model).
- Concise: 3 to 20 lines, focused on the core technique.
- Do NOT add import statements.
- Use def/class only when the technique genuinely needs a function.
- Do NOT include markdown fences or explanations.
- Do NOT hardcode competition-specific file paths.
- Start with the most distinctive line first.

User.

Generate a Python snippet for the following ML technique node.

Technique: [TECHNIQUE NAME]
Category: [CATEGORY]
Description: [TECHNIQUE DESCRIPTION]
Hyperparameters: [TECHNIQUE METADATA]
Existing code: [EXISTING CODE — low quality, improve or replace]

Write ONLY the Python code, no imports, no markdown fences.

A.10 Code-Snippet Review
Passes every node's snippet through a final KEEP-or-REWRITE pass with strong KEEP bias.

System.

You are reviewing code for a machine learning knowledge graph. Each node represents one technique (e.g., "pca", "stratified-kfold"). The code will be embedded by CodeT5+ using mean pooling over 512 tokens for retrieval.

Embedding-aware rules:

1. Do NOT add import statements. Imports waste token budget and make all nodes look similar in embedding space.
2. Do NOT wrap simple code in a function just to wrap it. A clear one-liner like `skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)` is MORE distinctive for retrieval than a generic wrapper. Use `def` only when the technique genuinely needs a multi-step routine.
3. Lead with the most distinctive code. The first tokens matter most for retrieval; do not open with boilerplate.
4. Keep it concise. Aim for 2 to 15 lines. Every token should carry information about THIS specific technique.

Default to KEEP. Only REWRITE when the code has a serious problem in one of these categories:
- Empty, commented out, shell command, or non-Python.
- Implements a DIFFERENT technique than described.
- Entirely competition-specific (hardcoded paths/columns, no reusable logic).
- Bloated beyond recognition (>2000 chars of irrelevant code).
- Just a config value with no API call (e.g., `NFOLDS=5` alone).

Write ONLY improved Python code when rewriting. No explanations, no fences.

User.

Technique: [TECHNIQUE NAME]
Category: [CATEGORY]
Description: [TECHNIQUE DESCRIPTION]
Hyperparameters: [TECHNIQUE METADATA]

Current code:
[EXISTING CODE]

Reply "KEEP" if the code acceptably illustrates the technique — even imperfect code that clearly shows the technique is fine. Reply "REWRITE" followed by improved code only if one of the problem categories applies.

Figure 10: Prompts for code-snippet quality control, including snippet selection, generation, and review.