Talking Trees: Reasoning-Assisted Induction of Decision Trees for Tabular Data

George Yakushev[*2,1] **Alina Shutova**[*2,1] **Ivan Rubachev**[*1,2]

Renat Sergazinov[*1] **Artem Babenko**[1,2]

Abstract

Tabular foundation models are becoming increasingly popular for low-resource tabular problems. These models make up for small training datasets by pretraining on large volumes of synthetic data. The prior knowledge obtained via pretraining provides the exceptional performance, but the resulting model becomes a black box that is difficult to interpret and costly to inference. In this work, we explore an alternative strategy: using reasoning-capable LLMs to induce decision trees for small tabular datasets in agentic setup. We design a minimal set of tools for constructing, analyzing and manipulating decision trees. By using these tools, LLMs combine their prior knowledge with learning from data to create a lightweight decision tree that outperforms traditional CART on low-resource tabular problems. While a single decision tree does not outperform state-of-the-art black box models, it comes with a human-readable reasoning trace that can be checked for biases and data leaks. Furthermore, the reasoning-based LLM's creation process allows for additional human input: correcting biases or incorporating domain-specific intuition that is not captured in the data. While a single decision tree does not outperform state-of-the-art black box models, it comes with a human-readable reasoning trace that can be checked for biases and data leaks. Furthermore, the reasoning-based LLM's creation process allows for additional human input: correcting biases or incorporating domain-specific intuition that is not captured in the data.

1 Introduction

Tabular tasks often come with problem-specific context, such as domain meta-knowledge, data collection caveats, and application constraints such as fairness, safety, and regulatory requirements. This additional information is not always captured in the dataset, especially when the data is limited.

A standard ML strategy for problems with limited data is transfer learning: pre-train a model on large quantities of diverse relevant data, then adapt it through fine-tuning or in-context learning [1, 2, 3]. For tabular data, this strategy is implemented by the recent tabular foundational models, such as TabPFN [4], which currently represent the state-of-the-art on many low/medium-size benchmarks [5, 6]. By pretraining on large collections of synthetic or real tabular tasks, such models achieve strong performance in few-shot and small-to-medium scale data regimes, outperforming decision-tree based models, such as Random Forests [7] and GBDTs [8, 9]. However, in practice, tabular foundation models are effectively black-box predictors, which are also costly in terms of the compute [10, 5].

In this work, we explore a more interpretable and lightweight alternative: instead of tabular foundation models, we employ a strong agentic LLM to construct an explicit decision tree via chain-of-thought reasoning and tool use. This gives our approach several important advantages over black-box methods. First, the decision tree is fast and lightweight. Second, both the LLM reasoning and the tree can be

[*]Equal contribution. [1]Yandex [2]HSE University. Correspondence to: yakushev-ga@yandex-team.ru

AI for Tabular Data workshop at EurIPS 2025.

Table 1: **Comparison**. †Interpretability mainly at the *feature* level.

Method	Family	Uses Prior	Steerable	Interpretable	Light Inference
Agentic Tree (ours)	Agentic Tree	✓	✓	✓	✓
OCTree (2024)	FE Agent	✓	✓	✓†	✓
CAAFE (2023/24)	FE Agent	✓	✓	✓†	✓
LLM-FE (2025)	FE Agent	✓	✓	✓†	✓
LLMBoost (2025)	Booster	✓	✗	✗	✗
TabLLM (2022)	Direct LLM	✓	✗	✗	✗
Tabula-8B (2024)	Direct LLM	✓	✗	✗	✗
TabPFN (2022→)	Foundational ICL	✓	✗	✗	✓
TabICL / LIMIX (var.)	Foundational ICL	✓	✗	✗	✗

manually or automatically checked for biases or leaks [11, 12]. Further, the users can specify any additional problem-specific knowledge that is not captured in the training data through prompting. In our tests on a set of real-world data, we observe our LLM-constructed models outperform CART decision tree induction. Though individual agentic trees still do not outperform state-of-the-art black-box methods, they provide a competitive baseline which is also steerable, interpretable and fast.

2 Related Works

Recent tabular work splits into three lines. Agentic LLMs for feature engineering (OCTree [13], CAAFE [14], LLM-FE [15]) use language-model priors plus validation feedback to discover useful transformations; they often pair with a conventional downstream learner, improving accuracy but leaving the predictor itself opaque. LLMs as prediction machines (TabLLM [16], Tabula-8B [17], LLMBoost [18]) bypass feature learning and produce predictions directly (or ensemble them), yielding strong small-N performance but high inference cost and limited auditability. Foundational/ICL priors for tabular (TabPFN family [4]; TabICL [19], LIMIX [6] approaches) pretrain on vast synthetic tasks and adapt in-context, offering excellent data efficiency but remaining black-box. Our approach occupies a complementary point: an agentic LLM that induces a symbolic predictor—a decision tree—combining small-data performance gains over CART with interpretability, auditability, and steerability that others largely lack.

3 Agentic Tree Induction

We reformulate the process of learning a decision tree as an agentic task [20, 21], giving the LLM tools to expand, analyze and manipulate the decision tree. This way, the LLM agent does not generate the entire tree in one go, but gradually refines it in small increments. We hypothesize that this incremental process will allow LLMs to better harness their capacity for chain-of-thought reasoning by testing their hypotheses against data and correcting their own errors.

To enable this type of agentic tree construction, we design a set of tools for analyzing and modifying the tree. These tools are organized as a single-file[1] Python library designed around the `Tree` class, which can be created directly or converted from `scikit-learn` trees [25]. We implement the agent using `smolagents.CodeAgent` [26], limiting execution to 20 steps. The model receives train and validation data, but not the test set. We provide detailed setup description in Appenidx A.

Safety and human input. LLM agents raise safety concerns regarding potential misuse [27]. In high-stakes settings, the tree-constructing agent can be fully air-gapped [28, 29], and reasoning traces can be inspected to catch issues such as data leaks or undesirable biases [11]. Conversely, using general-purpose LLM agents enables incorporating informal domain knowledge through prompts—from explaining physical models to adjusting for known data biases. Section 4.2 explores two such use cases.

4 Experiments

To recap, our main motivation for agentic decision tree learning is not to compete with state-of-the-art black-box models, but to attain a lightweight, interpretable and flexible learner that can take into

[1] See `editable_tree.py` at `https://github.com/yandex-research/TalkingTrees`

account additional inputs. To that end, we organize this section as follows: we evaluate LLM-constructed trees in section 4.1 and highlight how additional instructions can be incorporated in Section 4.2. In addition, we also vary the LLM backbone in Appendix C, perform tool ablation in Appendix D, and analyze LLM reasoning traces in Appenidx E.

4.1 Primary Evaluation

Table 2: Evaluating agent-constructed decision trees and advanced tabular methods on low-resource tabular datasets. The datasets are split into three groups by task type. Gray values are adjusted standard deviations over 5 splits (TabArena v0.1).

Method \ Dataset	Fitness	Marketing	CreditG	Diabetes	Customer	QSARbio	Hazelnut	Blood	Anneal	Maternal	Phishing	MIC	AirfoilSN	UsedFiat500	Concrete	Insurance	QSARfish
	Classification: Binary (ROC AUC↑)								Multiclass (LogLoss↓)				Regression (RMSE↓)				
CART (default)	0.615 ±0.015	0.672 ±0.021	0.615 ±0.029	0.676 ±0.038	0.549 ±0.017	0.781 ±0.017	0.818 ±0.017	0.554 ±0.041	0.529 ±0.277	5.44 ±1.29	3.82 ±0.42	6.89 ±0.54	2.957 ±0.098	1003.1 ±14.85	7.67 ±0.30	6380.9 ±427.3	1.299 ±0.034
CART (tuned)	0.800 ±0.021	0.799 ±0.026	0.700 ±0.033	0.779 ±0.026	0.698 ±0.011	0.852 ±0.025	0.895 ±0.014	0.698 ±0.028	0.321 ±0.067	0.849 ±0.141	0.674 ±0.197	0.538 ±0.025	2.988 ±0.134	789.8 ±31.59	7.65 ±0.37	4494.3 ±167.3	1.043 ±0.047
Ours (single tree)	0.818 ±0.017	0.820 ±0.032	0.717 ±0.011	0.799 ±0.029	0.717 ±0.018	0.854 ±0.018	0.904 ±0.013	0.716 ±0.011	0.138 ±0.108	0.739 ±0.149	0.476 ±0.202	0.467 ±0.024	2.956 ±0.180	790.9 ±41.45	7.28 ±0.34	4439.8 ±159.1	1.018 ±0.041
XGBoost (TabArena)	0.814 ±0.016	0.907 ±0.017	0.788 ±0.011	0.839 ±0.016	0.738 ±0.017	0.934 ±0.018	0.975 ±0.005	0.719 ±0.035	0.023 ±0.017	0.423 ±0.041	0.255 ±0.016	0.446 ±0.019	1.406 ±0.091	745.1 ±23.63	4.17 ±0.08	4580.4 ±517.7	0.870 ±0.052
TabM (TabArena)	0.822 ±0.013	0.917 ±0.013	0.792 ±0.021	0.838 ±0.015	0.738 ±0.010	0.935 ±0.019	0.982 ±0.005	0.723 ±0.032	0.020 ±0.015	0.450 ±0.044	0.240 ±0.033	0.438 ±0.015	1.114 ±0.093	753.6 ±22.5	4.25 ±0.35	4585.6 ±472.2	0.886 ±0.037
TabPFN v2 (TabArena)	0.824 ±0.012	0.919 ±0.016	0.761 ±0.014	0.846 ±0.007	0.738 ±0.013	0.934 ±0.019	0.988 ±0.003	0.742 ±0.033	0.021 ±0.010	0.413 ±0.043	0.223 ±0.025	0.445 ±0.025	1.045 ±0.073	736.1 ±31.7	4.00 ±0.06	4717.3 ±553.0	0.854 ±0.063
TabPFN v2 + Ours	0.828 ±0.017	0.917 ±0.020	0.792 ±0.013	0.843 ±0.022	0.738 ±0.015	0.937 ±0.011	0.990 ±0.003	0.740 ±0.023	0.014 ±0.004	0.426 ±0.055	0.218 ±0.028	0.445 ±0.023	1.029 ±0.092	722.8 ±21.9	4.03 ±0.07	4620.0 ±596.9	0.849 ±0.058

First, we evaluate on a range of low-resource tabular problems. Similarly to previous work [4], we select tabular classification and regression problems from the OpenML platform [30, 31]. Namely, we select all 17 supervised datasets from the `tabarena-v0.1` [10] benchmark that contain no more than 2500 samples. This includes 8 binary classification, 4 multiclass classification and 5 regression datasets. Following the benchmark, we report ROC AUC for binary classification, LogLoss for multiclass, and RMSE for regression problems. We provide full dataset names and identifiers in Appendix B.

We summarize our findings in Table 2 (top panel). Overall, the LLM-made decision trees surpass CART decision trees (default and tuned) in all but one dataset. We also report a number of advanced (but not interpretable) baseline methods, including XGBoost [9] (tuned GBDT), TabM [32] (advanced NN ensemble) and TabPFN v2 [4, 33] (tabular foundation model) to better illustrate the gap between simple interpretable models and state-of-the-art. In addition, we also evaluate agentic decision trees on top of TabPFN v2. In this setting, the LLM agent is tasked to learn the residual of TabPFN v2. The results in Table 2 (bottom panel) show that black-box methods still outperform individual decision trees on most problems, but the agentic tree learning can significantly reduce that gap. Additionally, using agentic decision trees to correct TabPFN predictions leads to somewhat better performance in most cases. While this two-staged pipeline is not interpretable (due to TabPFN), it can be useful for tasks where the agentic decision tree can correct biases or incorporate additional inputs.

4.2 Additional Human Inputs

One of the unique advantages of LLM-based tree construction is that the task can be defined informally, through the prompt. This allows the user to specify additional requirements that can be difficult or impossible to formalize for traditional decision trees and even for more advanced tabular methods.

In this section, we consider two such scenarios: i) **fairness:** correcting the bias in the data and ii) **informal feature definition**, where the system was just given access to a new feature that is not yet present in historical data, but there is an informal description of how it affects the task.

4.2.1 Fairness

Fairness has become an important concern in machine learning, as predictive models are increasingly deployed in socially sensitive domains [34, 35, 36]. The research community proposed several criteria of fairness, such as group-based and individual statistical parity [34, 37], and practical algorithms for incorporating fairness constraints into classifiers [38, 39]. We only conduct initial exploration of this problem, evaluating how LLM agents control for fairness when building their decision tree *if we prompt them to do so*, without any explicit algorithmic modifications.

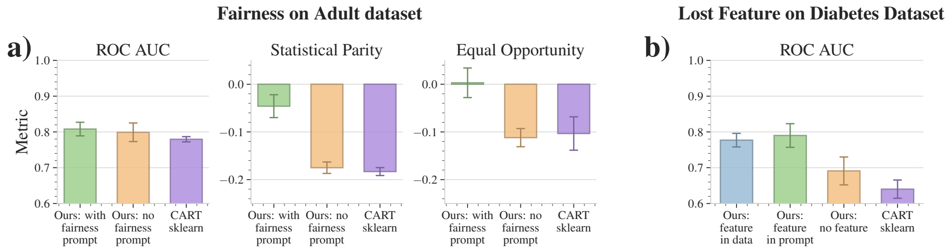

Figure 1: **a)** Fairness evaluation on the Adult dataset across three setups: LLM-built trees with and without the fairness prompt and the sklearn baseline. **b)** Training with experiment on the Diabetes dataset: performance of trees trained with and without access to the «Glucose» feature. Both experiments use GPT-5 backbone the setup from Section 4.1.

We evaluate fairness on the `Adult` binary classification dataset [40] known to contain gender-based bias[2]. We report two widely used fairness metrics: *Statistical Parity* [34] and *Equal Opportunity* [42]. If $Y \in \{0, 1\}$ is the classification target, $\hat{Y}$ is the model prediction, and $A \in \{a, b\}$ is the protected attribute (gender), then the Statistical Parity Difference (SPD) is: $P(\hat{Y} = 1 \mid A = a) - P(\hat{Y} = 1 \mid A = b)$. In turn, the Equal Opportunity Difference (EOD) is: $P(\hat{Y} = 1 \mid A = a, Y = 1) - P(\hat{Y} = 1 \mid A = b, Y = 1)$. A perfectly fair model would yield SPD = EOD = 0.

We compare three configurations: **(1) Ours: with fairness prompt** – LLM-built tree with prompt requesting gender-neutrality (Appendix F); **(2) Ours: no fairness prompt** – LLM-built tree without fairness instructions (Section 4.1); **(3) CART sklearn** – `scikit-learn` decision tree with same tuning protocol. We report our results in Figure 1 (a). To summarize, an informal request to adjust for gender bias results in significantly reduced SPD and EOD. As expected, the bias-adjusted tree also has somewhat lower AUC, but still outperforms a `sklearn` based tree.

4.2.2 Informal Feature Description

We consider a setup where a model has access to an additional feature during deployment but not in training data. This can occur, for example, when: **(1)** newly introduced medical tests lack historic patient data for model training, or **(2)** legal requirements prevent storing sensitive customer information that can be used during interaction.

In such cases, important features are missing from training data, but informal domain knowledge exists about their utility. We feed this knowledge in the LLM agent's prompt and evaluate the final tree on a dataset with the extra feature. We use the `Diabetes` dataset, hiding `Glucose` from training data.

We compare four setups: **(1) Ours: feature in data** – LLM-built tree with feature in training data; **(2) Ours: feature in prompt** – LLM-built tree without the feature but with a prompt (Appendix F) describing its absence and relevance; **(3) Ours: no feature** – without feature in prompt or data; **(4) CART sklearn** – decision tree without the feature. Figure 1 (b) shows both **Feature in prompt** and **Feature in data** achieve similar performance, clearly outperforming trees without the feature. Small datasets combined with LLM priors enable effective compensation for missing features, demonstrating potential for incorporating human input or model priors in such settings.

5 Conclusion and Discussion

In this working paper, we have shown that LLM agents can construct and improve decision trees without specialized training. Our results show that the LLM-guided decision trees can effectively combine different sources of knowledge: the tabular data itself, feature description and informal domain knowledge. While individual LLM-induced trees do not outperform black-box foundation models, they offer a lightweight and interpretable alternative and can incorporate additional task-specific demands.

[2] `Adult` also has known limitations [41] that can be important for dedicated fairness studies, but it is sufficient for our simple analysis. See Appendix B for more details.

References

[1] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[4] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*, 2022.

[5] Ivan Rubachev, Akim Kotelnikov, Nikolay Kartashev, and Artem Babenko. On finetuning tabular foundation models. *arXiv preprint arXiv:2506.08982*, 2025.

[6] Xingxuan Zhang, Gang Ren, Han Yu, Hao Yuan, Hui Wang, Jiansheng Li, Jiayun Wu, Lang Mo, Li Mao, Mingchao Hao, et al. Limix: Unleashing structured-data modeling capability for generalist intelligence. *arXiv preprint arXiv:2509.03505*, 2025.

[7] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[8] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[9] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[10] Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Mutalik Desai, David Salinas, and Frank Hutter. Tabarena: A living benchmark for machine learning on tabular data. *arXiv preprint arXiv:2506.16791*, 2025.

[11] Tomek Korbak, Mikita Balesni, Elizabeth Barnes, Yoshua Bengio, Joe Benton, Joseph Bloom, Mark Chen, Alan Cooney, Allan Dafoe, Anca Dragan, Scott Emmons, Owain Evans, David Farhi, Ryan Greenblatt, Dan Hendrycks, Marius Hobbhahn, Evan Hubinger, Geoffrey Irving, Erik Jenner, Daniel Kokotajlo, Victoria Krakovna, Shane Legg, David Lindner, David Luan, Aleksander Mądry, Julian Michael, Neel Nanda, Dave Orr, Jakub Pachocki, Ethan Perez, Mary Phuong, Fabien Roger, Joshua Saxe, Buck Shlegeris, Martín Soto, Eric Steinberger, Jasmine Wang, and Wojciech Zaremba. Chain of thought monitorability: A new and fragile opportunity for ai safety. *arXiv preprint arXiv:2507.11473*, 2025.

[12] Benjamin Arnav, Pablo Bernabeu Perez, Nathan Helm-Burger, Tim Kostolansky, Hannes Whittingham, and Mary Phuong. Cot red-handed: Stress testing chain-of-thought monitoring. *ArXiv*, abs/2505.23575, 2025.

[13] Jaehyun Nam, Kyuyoung Kim, Seunghyuk Oh, Jihoon Tack, Jaehyung Kim, and Jinwoo Shin. Optimized feature generation for tabular data via llms with decision tree reasoning. *Advances in Neural Information Processing Systems*, 37:92352–92380, 2024.

[14] Noah Hollmann, Fabian Müller, and Frank Hutter. Context-aware automated feature engineering (caafe). *arXiv preprint*, (2305.03403), 2023.

[15] Nikhil Abhyankar, Parshin Shojaee, and Chandan K Reddy. Llm-fe: Automated feature engineering for tabular data with llms as evolutionary optimizers. *arXiv preprint arXiv:2503.14434*, 2025.

[16] Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. Tabllm: Few-shot classification of tabular data with large language models. In *International conference on artificial intelligence and statistics*, pages 5549–5581. PMLR, 2023.

[17] Josh Gardner, Juan C Perdomo, and Ludwig Schmidt. Large scale transfer learning for tabular data via language modeling. *Advances in Neural Information Processing Systems*, 37:45155–45205, 2024.

[18] Mayuka Jayawardhana, Samuel Dooley, Valeriia Cherepanova, Andrew Gordon Wilson, Frank Hutter, Colin White, Tom Goldstein, Micah Goldblum, et al. Transformers boost the performance of decision trees on tabular data across sample sizes. *arXiv preprint arXiv:2502.02672*, 2025.

[19] QU Jingang, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. Tabicl: A tabular foundation model for in-context learning on large data. In *Forty-second International Conference on Machine Learning*.

[20] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761, 2023.

[21] Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789, 2023.

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

[23] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *ArXiv*, abs/2205.11916, 2022.

[24] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alexander J. Smola. Automatic chain of thought prompting in large language models. *ArXiv*, abs/2210.03493, 2022.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 'smolagents': a smol library to build great agentic systems. `https://github.com/huggingface/smolagents`, 2025.

[27] Feng He, Tianqing Zhu, Dayong Ye, Bo Liu, Wanlei Zhou, and Philip S. Yu. The emerged security and privacy of llm agent: A survey with case studies. *arXiv preprint arXiv:2407.19354*, 2024.

[28] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. *arXiv preprint arXiv:2309.15817*, 2023.

[29] Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, Peter Kairouz, Marco Gruteser, Sewoong Oh, Borja Balle, and Daniel Ramage. Airgapagent: Protecting privacy-conscious conversational agents. *arXiv preprint arXiv:2405.05175*, 2024.

[30] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15:49–60, 2013.

[31] Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter. Openml-python: an extensible python api for openml. *Journal of Machine Learning Research*, 22(100):1–5, 2021.

[32] Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. Tabm: Advancing tabular deep learning with parameter-efficient ensembling. In *Proceedings of the 2025 International Conference on Learning Representations (ICLR 2025)*, 2025. arXiv:2410.24210 [cs.LG], version v3.

[33] Shi Bin Hoo, Samuel Müller, David Salinas, and Frank Hutter. The tabular foundation model tabpfn outperforms specialized time series forecasting models based on simple features. *arXiv preprint arXiv:2501.02945*, 2025.

[34] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS)*, pages 214–226, 2012.

[35] Alexandra Chouldechova and Aaron Roth. The frontiers of fairness in machine learning. *arXiv preprint arXiv:1810.08810*, 2018.

[36] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning: Limitations and Opportunities*. MIT Press, 2023.

[37] Shira Mitchell, Eric Potash, Solon Barocas, Alexander D'Amour, and Kristian Lum. Prediction-based decisions and fairness: A catalogue of choices, assumptions, and definitions. *arXiv preprint arXiv:1811.07867*, 2018.

[38] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 60–69, 2018.

[39] Sorelle A. Friedler, Carlos Scheidegger, Suresh Venkatasubramanian, Sonam Choudhary, Evan P. Hamilton, and Derek Roth. A comparative study of fairness-enhancing interventions in machine learning. *arXiv preprint arXiv:1802.04422*, 2018.

[40] Ron Kohavi and Barry Becker. Uci adult data set. UCI Machine Learning Repository, 1996. https://archive.ics.uci.edu/ml/datasets/adult.

[41] Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring adult: New datasets for fair machine learning. In *NeurIPS Datasets and Benchmarks Track*, 2021.

[42] Moritz Hardt, Eric Price, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[43] Python Core Team. *Python: A dynamic, open source programming language*. Python Software Foundation, 2019.

[44] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[45] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and the SciPy 1.0 Contributors. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3):261–272, 2020.

[46] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[47] Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *arXiv preprint arXiv:2305.04388*, 2023.

[48] Barry Becker and Ron Kohavi. Adult [dataset]. UCI Machine Learning Repository, 1996. https://archive.ics.uci.edu/ml/datasets/adult.

A Additional Method Details

The overall schematic illustration of the proposed agentic tree induction methods is illustrated in Figure 2. Below we provide more detail regarding actual implementation. For full reproducibility we also make the relevant code available.

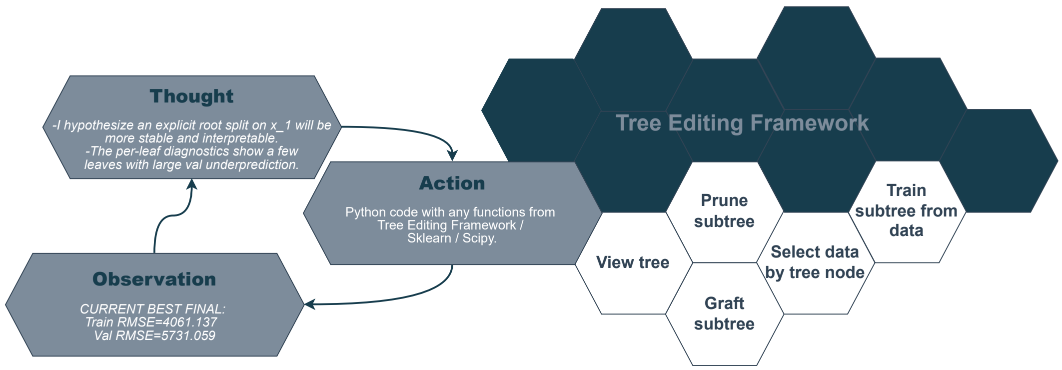

Figure 2: The informal summary of our approach. We prompt an LLM agent to construct a decision tree in a thought-action-observation cycle (left). During the action phase, the agent can use a tree editing framework (right) with tools for analyzing and modifying trees. The agent can combine these tools in Python and print arbitrary diagnostics for the observation phase.

We reformulate the process of learning a decision tree as an agentic task [20, 21], giving the LLM tools to expand, analyze and manipulate the decision tree. This way, the LLM agent does not generate the entire tree in one go, but gradually refines it in small increments. We hypothesize that this incremental process will allow LLMs to better harness their capacity for chain-of-thought reasoning by testing their hypotheses against data and correcting their own errors.

To enable this type of agentic tree construction, we design a set of tools for analyzing and modifying the tree. These tools are organized as a single-file[3] Python library designed around the `Tree` class. The tree can be created as follows:

```
Tree("feat1<=0.5",       # if feat1 <= 0.5:
     le=3.5,             #   return 3.5
     gt=Tree("feat2<=9", # elif feat2 <= 9:
             le=-2,      #   return -2
             gt=+2))     # else: return 2
```

or converted from a `scikit-learn` tree[4] [25].

We give the LLM agent access to the following tools for analyzing and modifying the tree:

1. **View tree:** `print(tree)` displays the tree or chosen subtree structure as text. The printed tree includes node IDs that can be used via `tree.find_node(id)`.
2. **Prune subtree:** converts a tree node into a leaf. This can be done manually, e.g. `tree.gt.prune()`, or by node ID `tree.find_node(id).prune()`.
3. **Select data:** run the tree on dataset X, select samples that pass through a given node ID: `tree.get_data_indices_for_node(id, X)`
4. **Graft subtree:** replace the specified node (by ID) in the current tree with the new (manual or learned) subtree: `tree.replace_subtree(id, new_subtree)`

These tools were designed to be composable, allowing the agent to implement more complex tree manipulations:

```
sktree = DecisionTreeRegressor(max_depth=6)
sktree.fit(X_train, y_train)
tree = Tree.from_sklearn(sktree)
```

[3] See `editable_tree.py` in the supplemetnary code. The library was "vibe-coded" with GPT-5 via `chatgpt.com`.

[4] `sklearn` stores trees as hard-to-edit packed data structures.

```python
print(tree)  # observe node IDs
tree.find_node(42).prune()
# manual tree editing
node = tree.find_node(43)
node.feature = "sepal_length"
node.threshold = 6.5
node.gt = Tree("width<=5", le=-3, gt=+3)
node.le = Tree(value=0) # leaf
# semi-automated manipulation
ix = tree.get_data_indices_for_node(
    id=44, X=X_train
)
features = my_features(X_train.iloc[ix])
subtree = DecisionTreeRegressor(max_depth=2))
subtree.fit(
    X=X_train.iloc[ix][features],
    y=y_train[ix]
)
tree.replace_subtree(
    44, Tree.from_sklearn(new_subtree)
)
```

Our framework also implements several smaller utility tools, such as accessing the node properties (feature, threshold), re-indexing node IDs after edits and refitting tree leaves from data, and common aliases, all included in the supplementary code. In addition to these tools, the python-based LLM agent has access to NumPy [44], SciPy [45], pandas [46], scikit-learn [25] and common python builtins such as `math`. We found that LLMs often define their own auxiliary functions depending on the task, such as selecting "promising" leaves for expansion or fitting a subtree without outliers.

The LLM agent starts with a system prompt[5] that explains the thought-action-observation cycle and describes the available tools. The system prompt also describes the agent's overall task (i.e. construct and iteratively improve a decision tree with data) and specifies the desired output format.

When building a tree for a tabular dataset, the agent receives the training and validation subsets, *but it has no access to the test data*. Additionally, we provide the LLM with a short description of the input features and the objective.

We implement the agent using `smolagents.CodeAgent` [26], with a custom environment and system prompt. We inject the data into the agent's environment as pre-defined variables (e.g. `X_train`) and expect it to return the final decision `Tree`. Aside from that, we use the default `smolagents` hyperparameters up to 20 thought-code-observation cycles. The entire process takes around 3-10 minutes and $\leq$0.3$ API costs per tree even for state-of-the-art LLMs (e.g. GPT-5) and can be parallelized for ensembling. In theory, an agent can fail to return the tree in the correct format, but we found that modern LLMs almost always produce valid decision trees, albeit of varying quality (see Section D).

Safety matters. More importantly, the LLM agents can potentially misuse this free-form communication to malicious ends, which raises important safety concerns [27]. In high-stakes settings, the tree-constructing agent can[6] be fully air-gapped [28, 29]. More specifically, the system's Python environment can run in an isolated container with no network. In addition, the output `Tree` needs to be protected from potential malware injection. This can be achieved by serializing the tree in a format that does not allow arbitrary code execution, e.g. only supporting the tree structure and limited mathematical expressions for feature engineering. With a trusted (or local) LLM inference provider, such air-gapped system ensures that the agent does not leak the data and does not inject malicious code into the final model.

Another partial remedy to combat problematic behavior in LLM is to inspect the reasoning traces produced by the model during tree construction and refinement [11]. This kind of manual inspection can catch issues such as data leaks exploitation, undesirable biases, or specification gaming for the target metric. However, it must be noted that the lack of such problematic behaviors in model "thoughts" is not a strong guarantee as LLMs do not always articulate their biases during reasoning [47].

[5]The full system prompt is included in the supplementary code.

[6]Note that our implementation is not fully air-gapped by default, but it can be done using existing `smolagents` components.

Additional human input. Using general-purpose LLM agents comes with safety concerns, but it also introduces an important new capability: harnessing informal domain knowledge (as prompt) in addition to the training data. This covers a broad range of use cases from explaining the underlying physical model for engineering problems to sharing intuition or adjusting for known problems in the data. In Section 4.2, we explore two such use cases: correcting for a known bias and using a new feature (e.g. new test for medical diagnosis) not available in historical training data.

B Evaluation Dataset Information

The main evaluation uses the following datasets (with OpenML task IDs).

- **Fitness** - `Fitness_Club` (ID 363671), 1500 samples
- **Marketing** - `Marketing_Campaign` (ID 363684), 2240 samples
- **CreditG** - `credit-g` (ID 363626), 1000 samples
- **Diabetes** - `diabetes` (ID 363629), 768 samples
- **Customer** - `Is-this-a-good-customer` (ID 363682), 1723 samples
- **QSARbio** - `qsar-biodeg` (ID 363696), 1054 samples
- **Hazelnut** - `hazelnut-spread-contaminant-detection` (ID 363674), 2400 samples
- **Blood** - `blood-transfusion-service-center` (ID 363621), 748 samples
- **Anneal** - `anneal` (ID 363614), 898 samples
- **Maternal** - `maternal_health_risk` (ID 363685), 1014 samples
- **Phishing** - `website_phishing` (ID 363707), 1353 samples
- **MIC** - `MIC` (ID 363711), 1699 samples
- **AirfoilSN** - `airfoil_self_noise` (ID 363612), 1503 samples
- **UsedFiat500** - `Another-Dataset-on-used-Fiat-500` (ID 363615), 1538 samples
- **Concrete** - `concrete_compressive_strength` (ID 363625), 1030 samples
- **Insurance** - `healthcare_insurance_expenses` (ID 363675), 1338 samples
- **QSARfish** - `QSAR_fish_toxicity` (ID 363698), 907 samples

B.1 Human Input

Additionally, we use the `Adult` dataset [48] for the fairness experiments, as it is the standard benchmark for evaluating group fairness with respect to gender, and the `Diabetes` dataset for the missing feature experiments, since it contains the clinically important `Glucose` variable that we deliberately removed from train split in relevant setups.

We chose the `Adult` dataset despite its known limitations [41] such as class imbalance, demographic biases, and potential label noise, because it is a widely used benchmark in fairness research. Its prevalence enables direct comparison of our fairness metrics with prior work, and its protected attribute structure (e.g., gender) is well suited for controlled experiments on fairness.

C LLM Backbone Ablation

Next, we take a subset of datasets from the previous section to perform a more detailed comparison of different backbone LLMs. For clarity, we only vary the base LLM used within our agent while keeping the agentic framework, toolset, and prompts fixed[7]. We evaluate two closed-source models (GPT-5, Gemini 2.5 Pro) and three open-source models (GLM-4.5, Kimi K2 Instruct, DeepSeek R1-0528). For each dataset, we run five random seeds and report the mean and the adjusted standard deviation in Table 3.

[7]Concretely, only the reasoning model is swapped, while the following remain invariant across conditions: identical system and user prompts, the same minimal tree-manipulation tools (split proposal, split scoring, node expansion, pruning, and cross-validated evaluation), search budgets, and the same evaluation protocol.

Table 3: Comparison of base LLM models on selected datasets with the same evaluation setup as in Section 4.1.

Method \ Dataset	Fitness AUC↑	Anneal LogL↓	AirfoilSN RMSE↓
GPT-5	0.818 ±0.017	0.138 ±0.108	2.956 ±0.180
Gemini-2.5-pro	0.800 ±0.011	0.624 ±0.552	4.004 ±0.168
GLM 4.5	0.799 ±0.015	0.224 ±0.113	3.620 ±0.394
Kimi K2 Instruct	0.808 ±0.014	0.161 ±0.056	3.764 ±0.505
DeepSeek R1 0528	0.797 ±0.011	0.410 ±0.32	3.874 ±0.733

Table 4: Additional evaluations of agent-constructed tabular models on low-resource tabular datasets with different tools and environment restrictions.

Method \ Dataset	Fitness AUC↑	Anneal LogL↓	AirfoilSN RMSE↓
Main method	0.818 ±0.017	0.138 ±0.108	2.956 ±0.180
Unlimited (not tree-only)	0.807 ±0.006	0.060 ±0.056	1.621 ±0.094
Agent-sklearn trees	0.795 ±0.014	0.257 ±0.230	2.943 ±0.705
Main w/o description	0.813 ±0.011	0.125 ±0.043	3.508 ±0.438

On more challenging or low-signal tasks (Anneal and AirfoilSN), we observe the greatest variability, with GPT-5 achieving up to 30%–40% higher performance. Prior results from [10] and Table 2 confirm that these datasets exhibit sharp performance jumps across baseline families: from shallow learners such as KNN and linear models (Anneal: logloss ≈ 0.5, AirfoilSN: RMSE ≈ 3), to default gradient-boosted models (Anneal: logloss ≈ 0.02, AirfoilSN: RMSE ≈ 1.4), to state-of-the-art neural models (Anneal: logloss ≈ 0.01, AirfoilSN: RMSE ≈ 1.07). On the relatively easier Fitness dataset,the tool-augmented procedure appears to saturate performance, with model choice contributing marginally compared to the variance between train/test splits. This is unsurprising: even tuned KNN and linear models establish a stronger baseline on this task [10].

D Tool Ablation Analysis

In this section, we investigate the impact of the agent's operational boundaries and input information on final performance. We conduct three experiments: first, we grant the LLM unrestricted access to the entire scikit-learn library, challenging it to construct advanced model ensembles not limited to decision trees. Second, we replicate the tree-building experiments from Table 2 but using only scikit-learn frameworks, i.e. without the tree editing framework from Section 3. Finally, we evaluate the importance of supplementary dataset metadata provided to the LLM.

We summarize our results in Table 4. Unsurprisingly, the unrestricted LLM agent can outperform individual trees on harder datasets (Anneal, AirfoilSN) by using ensembles. However, it still fails to compete with advanced black-box methods in Table 2. In turn, the single tree agent without our editing tools consistently underperforms across 3 datasets, exhibiting very high standard deviation on AirfoilSN and Anneal tasks. In other terms, the agent utilizing our Tree Editing Framework demonstrably yields more stable and reliable outcomes. Finally, the impact of dataset metadata (description and feature names) varies between tasks. We found that giving the LLM access to informal description significantly improves some datasets (AirfoilSN), but has much smaller effect on others. More specifically, it slightly improves model on Fitness but worsens it on Anneal.

E Agent Analysis

In this section, we analyze the tree construction process in more detail to better understand current agent behavior and inform potential future improvements. We analyze the code generated on during each "action" phase, grouping function calls into categories.We categorize the agent's actions into the following five groups by their operational target (the model vs. the data) and their purpose (modification vs. analysis): 1) **tree modification** for training or altering the model; 2) **tree analysis** for model introspection; 3) **data analysis** for dataset-level exploration; 4) **data modification** for data cleaning, feature engineering & similar; and 5) **builtins** for supporting functions.

We illustrate our findings in Figure 3. The distribution reveals two broad regularities: 1) exploratory analysis and feature engineering together account for roughly half of all calls, and 2) generic helper/builtins contribute a comparable share, while direct tree edits (grafting/pruning/growing)

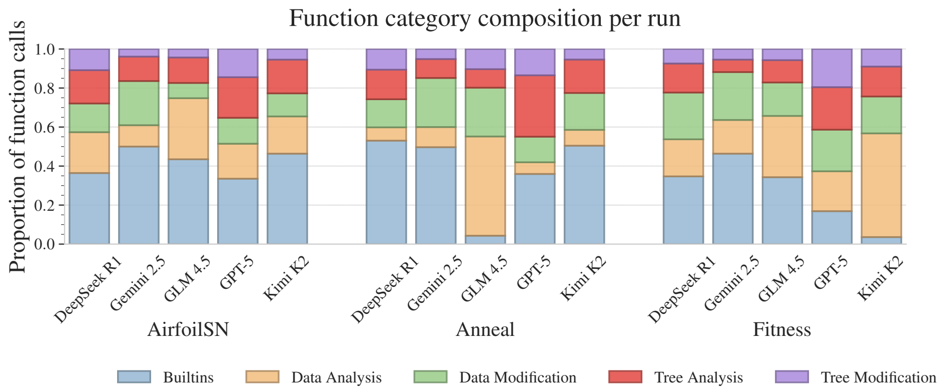

Figure 3: Tool call distribution across LLM backbones and datasets categorized by functionality.

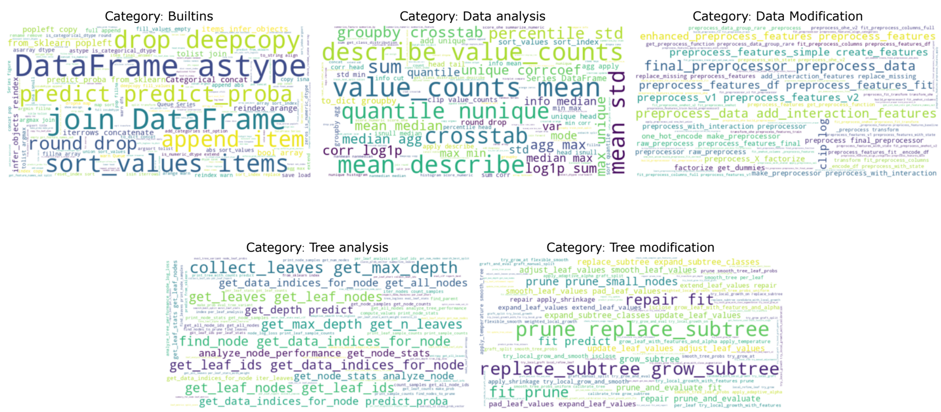

Figure 4: Word cloud of function calls by category.

remain a small tail, typically under 10%. In other words, the LLM agents typically performs a lot of analysis and testing before every edit. We also observe substantial model- and dataset-specific variability. GPT-5 exhibits the most balanced mix, with no single family dominating across tasks. By contrast, Gemini-2.5 pro – the weakest model in this specific setting – leans heavily on feature engineering and helper utilities, and consistently deprioritizes tree construction, with only about 5% of its calls devoted to explicit tree modifications. GLM-4.5, Kimi K2 Instruct, and DeepSeek R1 (0528) similarly skew toward general data-analysis operations. For context, we also provide word clouds for each function category in Appendix **??**.

Linear regression analysis. To disentangle the factors driving performance, we conducted a multiple linear regression analysis. We fit a regression that predicts a unified error score (lower is better, 1-AUC for binary classification) based on the chosen LLM, the dataset, and the proportion of generated function calls from five categories. This approach allows us to quantify the contribution of each factor while holding the others constant. The results (Table 5) suggest two key insights. First, after controlling for differences in strategy, GPT5 emerges as the most capable model ($\beta = -1.55, p < 0.001$), indicating a significant performance advantage attributable to its innate skill. Second, the choice of coding strategy has a profound impact on outcomes. Relative to the baseline helper category, a higher proportion of data analysis calls is significantly associated with better performance ($\beta = -1.01, p < 0.001$). Conversely, relying heavily on tree modification ($\beta = +8.80, p < 0.001$) and tree analysis ($\beta = +7.99, p < 0.001$) appears detrimental.

These results highlight that while top-performing models like GPT-5 succeed due to their superior baseline capabilities, their performance can be further optimized by guiding them towards more

Table 5: Regression Analysis of LLM Performance Factors. Negative coefficients correspond to reducing error (better).

Variable	Coefficient↓	Std. Err.
Fixed Effects (vs. 'Fitness' + 'Deepseek R1 0528')		
Intercept	-2.011**	±0.098
Dataset: AirfoilSN	2.970**	±0.023
Dataset: Anneal	-0.302**	±0.022
Model: GLM 4.5	0.681**	±0.047
Model: GPT-5	-1.555**	±0.059
Model: Gemini 2.5 Pro	0.809**	±0.047
Model: Kimi K2 Instruct	0.262*	±0.033
Function Category Proportions (vs. 'Builtins' baseline)		
Data Analysis	-1.012*	±0.105
Data Modification	2.333*	±0.255
Tree Analysis	7.990**	±0.351
Tree Modification	8.799**	±0.557

Significance: ** $p<0.001$; * $p<0.01$

effective coding strategies. In particular, data analysis provides valuable information into the dataset which allows the agent to make better informed and precise tree surgeries. Note that this is only an preliminary analysis: in future, it would be interesting to consider non-code LLM outputs and expand this analysis to more datapoints.

To visualize the most representative functions within each category, we generate a word cloud for each group, as shown in Figure 4. The size of each function in the cloud corresponds to its total call frequency, and a qualitative review confirms the semantic coherence of the automated categorization.

Prompt for Agent Call Classification

```
Help categorize the following functions into the following categories:
1.  Tree modifications:  'tree_mod' (key)
2.  Tree analysis, visualization and debugging:  'tree_eda' (key)
3.  General feature engineering and transformations:  'feat_engineering' (key)
4.  General exploratory data analysis:  'eda' (key)
5.  Builtins:  'builtins' (key)
Category descriptions:
1.  Tree modifications:  any operation that changes or trains the tree.  Key-
words:  DecisionTreeClassifier, DecisionTreeRegressor, min_samples_split,
max_depth, prune, replace_subtree, grow_subtree, repair, min_samples_leaf,
max_features, min_impurity_decrease, min_weight_fraction_leaf, ccp_alpha,
max_leaf_nodes, min_samples_leaf, min_samples_split, max_depth, max_features,
max_leaf_nodes, min_impurity_decrease, min_samples_leaf, min_samples_split,
min_weight_fraction_leaf, random_state, ccp_alpha.
2.  Tree analysis:  introspection of trained tree(s) such as paths,
leaves, importances, surrogate views, and plots.  Keywords:  is_leaf,
get_data_indices_for_node, print, decision_path, feature_importances_, ex-
port_graphviz.
3.  General feature engineering and transformations:  any input transformation
before training/inference.  Independent of a specific trained tree.  Keywords:
OneHotEncoder, PolynomialFeatures, clip, log1p, sign, datetime64.
4.  General exploratory data analysis:  dataset-level profiling and exploration
not tied to a specific model such as distributions, correlations, missingness,
leakage checks, class balance.  Keyword:  percentile, std, mean, tsne, umap,
train_test_split.
5.  Builtins:  infrastructure that doesn't change data or models and isn't
analysis such as I/O, seeding, logging, timing, config, small data wrangling
helpers.  Keywords:  check_random_state, asarray, dtype, save, load.
```

```
As an output provide a json-formattable dictionary of the form: category:
[function1, function2]. Specfcially, your answer will be directly plugged in
to the ‘json.loads(YOUR_ANSWER)‘. Please, respect the formatting.
Here are the functions: ‘set(defined_functions)‘.
```

F Human Input Prompts

The full prompts for all setups are available in our repository. Below we show the exact additional instructions we provided to the LLM in our human input experiments. Each prompt is inserted as an additional text to the base setup and is designed to inject domain knowledge or constraints that a standard execution would not capture.

Prompt for Fairness Human Input

```
You should prioritize creating model that as gender-neutral as possible even if
it harms metrics.
```

The above directions were used in the fairness experiment (4.2.1) to explicitly guide the LLM toward balancing outcomes across gender groups.

Prompt for Lost Feature Human Input

```
We forgot to collect ‘Glucose‘ data in the training set. This feature is ac-
tually very important for predicting diabetes, but you cannot see it in train.
Your task is to create a decision tree that may rely on this feature in the
test set, even though you have no Glucose data available during training.
```

The second instruction set was used in the lost feature experiment (4.2.2) to test whether the LLM could integrate prior knowledge about the missing variable into the tree design.

G Tree Construction Examples

In this section we present example reasoning traces produced during the tree construction process. These traces illustrate the step-by-step decisions made by the LLM while building or modifying decision trees under different constraints. For all experiments we used GPT-5 as the underlying model. In several examples code has been removed for readability. The following excerpts are shown to highlight representative reasoning patterns rather than complete outputs.

G.1 Main Experiment Examples

As an illustration of the LLM's workflow, we provide a reasoning trace consisting solely of the model's intermediate thoughts during a single run of tree construction. No code is shown, but the trace remains internally consistent and reflects the step-by-step process followed in that run.

Trace example on `Fitness` dataset (redacted)

```
- Thought: I will start by inspecting the raw data: shapes, NaN counts, class
balance, and unique values for categorical features. Hypothesis: there are
NaNs in weight, and potentially unseen categories in validation (e.g., category
"-"), so I will need imputation and robust one-hot encoding.
- Thought: I will design a robust preprocessing pipeline: impute missing
weight with the train median and add a missing indicator; one-hot encode cat-
egorical features; ensure consistent columns between train/val. Then I will
train a few sensible baseline DecisionTreeClassifier variants (small set of hy-
perparameters, with class_weight balanced) and pick the best by validation AUC.
I will convert it to editable_tree.Tree and print structure to form hypotheses
for manual edits.
```

```
- Thought:  I will inspect the current best tree's leaves and internal nodes
by computing how many train/val samples route to each node and the mean tar-
get (attendance rate) within those nodes.  Hypothesis:  some large leaves may
still be heterogeneous (e.g., split further by days_before or time), offering
room for local grafting/growth to improve AUC. I will list all nodes, identify
leaves, print their stats, then propose targeted edits on the biggest/most
heterogeneous leaves.
- Thought:  Hypothesis:  within large leaves, days_before, time, or category may
provide additional separation to improve AUC. I will select the top-2 largest
leaves and try growing a small local subtree (max_depth=2, min_samples_leaf=20)
using a focused set of features, then graft it, evaluate AUC, and keep changes
only if validation AUC improves.
- Thought:  I will analyze univariate relationships to guide targeted edits.
Hypothesis:  days_before and time or category may add predictive power, espe-
cially within the lower-membership region (months_as_member <= 15.5).  I'll
compute univariate AUCs and attendance means by bins/levels.  Then I'll try
local growth at key internal nodes (ids 1, 10, 11, 13), training small subtrees
on the routed subsets and grafting them.  I'll keep any graft that improves
validation AUC.
- Thought:  I'll add a few interpretable engineered features that reflect plau-
sible behavior:  weekend class indicator (Sat/Sun), early booking (>=10 and
>=7 days), high weight (>=85 kg), new member (<=12 months), and intense class
(HIIT or Cycling).  Hypothesis:  these features will help the tree form cleaner
splits with the same complexity and may improve AUC. Then I'll retrain a tree
with the previously best hyperparameters, inspect AUC, and if better, convert
to editable_tree for potential small local refinements (focused threshold tun-
ing limited to weight-only splits on the right branch).
```

Next, we present several selected examples of thought–code pairs extracted out of their full experimental context. These examples serve to illustrate what the agent is actually doing to achieve the stated in thoughts goals in constructing decision trees.

Thought-Code Pair example of local graft

```
Thought:  I will select the top-2 largest leaves and try growing a small local
subtree (max_depth=2, min_samples_leaf=20) using a focused set of features,
then graft it.
Code:

def try_local_grow(
    tree, node_id, Xtr, ytr, Xva, yva,
    feat_cols, max_depth=2, min_samples_leaf=20
):
    ids_tr = tree.get_data_indices_for_node(node_id, Xtr)
    ids_va = tree.get_data_indices_for_node(node_id, Xva)
    if len(ids_tr) < 2*min_samples_leaf:
        return None, None, None  # not enough data to grow
    # Fit a small subtree using a subset of features
    clf = DecisionTreeClassifier(
        random_state=0, class_weight="balanced",
        max_depth=max_depth, min_samples_leaf=min_samples_leaf
    )
    clf.fit(Xtr.iloc[ids_tr][feat_cols], ytr[ids_tr])
    # Evaluate partial AUC for diagnostics
    part_auc = roc_auc_score(
        yva[ids_va], clf.predict_proba(Xva.iloc[ids_va][feat_cols])[:,1]
    ) if len(np.unique(yva[ids_va]))>1 else np.nan
    # Graft into a copy of the tree
    new_tree = deepcopy(tree)
    new_tree.replace_subtree(node_id, Tree.from_sklearn(clf))
    # Evaluate global AUC
    val_auc = roc_auc_score(yva, new_tree.predict(Xva))
    train_auc = roc_auc_score(ytr, new_tree.predict(Xtr))
```

```
    return new_tree, train_auc, val_auc

# Try local growth on top-2 leaves
candidates = []
for lid, ntr, nva, mtr, mva in leaf_stats_sorted[:2]:
    print(f"\nAttempt local growth at leaf id={lid} (train n={ntr}, val n={nva})")
    new_tree, tr_auc, va_auc = try_local_grow(
        etree, lid, Xtr, y_train, Xva, y_val,
        focused_feats, max_depth=2, min_samples_leaf=20
    )
    if new_tree is not None:
        print(
          f" Local growth result -> Train AUC={tr_auc:.4f}, Val AUC={va_auc:.4f}"
        )
        candidates.append((va_auc, tr_auc, lid, new_tree))
    else:
        print(" Not enough samples to grow here.")
```

Thought-Code Pair example of Leaf label smoothing

```
Thought:  I'll try three conservative, targeted tweaks and keep any that im-
prove or match validation AUC:
- Laplace-smooth leaf probabilities to reduce overconfident small leaves.  Hy-
pothesis:  small leaves on the right branch could be slightly overconfident;
smoothing might improve ranking.
Code:
```

```
def laplace_smooth_tree(tree_in, Xproc, y, alpha=1.0):
    t = deepcopy(tree_in)
    for leaf in get_leaf_nodes(t):
        idx = t.get_data_indices_for_node(leaf.id, Xproc)
        n = len(idx)
        if n == 0:
            continue
        pos = float(y[idx].sum())
        leaf.value = (pos + alpha) / (n + 2*alpha)
    return t

sm_tree = laplace_smooth_tree(best_tree, Xtr, y_train, alpha=1.0)
sm_tr_auc = roc_auc_score(y_train, sm_tree.predict(Xtr))
sm_va_auc = roc_auc_score(y_val, sm_tree.predict(Xva))
print(f"Laplace(alpha=1): train AUC={sm_tr_auc:.4f}, val AUC={sm_va_auc:.4f}")
candidate_tree = deepcopy(best_tree)
cand_tr_auc, cand_va_auc = base_tr_auc, base_va_auc

if sm_va_auc >= cand_va_auc:
    candidate_tree = deepcopy(sm_tree)
    cand_tr_auc, cand_va_auc = sm_tr_auc, sm_va_auc
    print("Adopted Laplace smoothing as candidate.")
```

G.2 TabPFN Correction Examples

Here we present examples of thought–code pairs extracted from the traces of the TabPFN Correction experiments.

TabPFN Correction Thought-Code pair example

```
Thought:  The numeric stump is close but slightly below baseline.  Hypothesis:
adding a second split on credit_history on the high amount_per_month side can
capture stable residual bias (critical account slightly underpredicted, exist-
ing paid duly slightly overpredicted).  I will:
```

```
- Manually build a 2-level tree:  root split on amount_per_month; right child
splits on credit_history (critical vs existing duly vs others).
Code:

# Helper to build a manual tree given threshold and leaf values
def build_tree(thresh, v_low, v_crit, v_exist, v_other):
    # Right branch: first check critical account,
    # else check existing duly, else other

    # One-hot column names (ensure they exist; if not, they will be zeros)
    crit_col = "credit_history_critical account / other credits existing"
    exist_col = "credit_history_existing credits paid duly till now"

    # Construct right subtree
    right = Tree(
        id=10, feature=crit_col, threshold=0.5,
        left=Tree(  # not critical
            id=11, feature=exist_col, threshold=0.5,
            # not existing duly -> "other" categories
            left=Tree(id=12, value=float(v_other)),
            # existing duly
            right=Tree(id=13, value=float(v_exist))
        ),
        right=Tree(id=14, value=float(v_crit)) # critical account
    )
    root = Tree(id=0, feature="amount_per_month", threshold=float(thresh),
        left=Tree(id=1, value=float(v_low)),
        right=right)
    return root

for t in thresholds:
    m_left, m_crit, m_exist, m_other = residual_means(t)
    for eta in etas:
        tree_cand = build_tree(
            t, eta*m_left, eta*m_crit, eta*m_exist, eta*m_other
        )

        # Evaluate AUCs when added to baseline
        ytr_pred = y_train_baseline + tree_cand.predict(Xtr_ch)
        yva_pred = y_val_baseline + tree_cand.predict(Xva_ch)
        tr_auc = roc_auc_score(y_train, ytr_pred)
        va_auc = roc_auc_score(y_val, yva_pred)
        results.append((
            t, eta, m_left, m_crit, m_exist,
            m_other, tr_auc, va_auc, tree_cand
        ))
        print(
            f"MANUAL depth2 t={t}, eta={eta}: "
            f"Train AUC={tr_auc:.6f} Val AUC={va_auc:.6f}"
        )
```

G.3 Fairness Example

To better illustrate how the LLM incorporates fairness instructions into the tree-building process, we show below a redacted internal reasoning trace on the `Adult` dataset. This is just one example of how the LLM can approach the task; other strategies are also possible, such as first training a tree on the full dataset (including the sex feature) and then pruning subtrees that heavily rely on it.

Fairness trace example on `Adult` dataset (redacted)

```
- Thought:  I will inspect the raw data to understand dtypes, missing values ,
class balance, and categorical levels.
```

- Thought: The current tree relies heavily on marital-status at the root, which is strongly correlated with gender outcomes via roles like Husband/Wife (even though “relationship” itself is excluded). To improve gender neutrality, I will first drop the relationship feature, then explicitly retrain while excluding both relationship and marital-status to reduce gender proxies. I will iteratively adjust depth/regularization to avoid rediscovering proxy splits.
- Thought: I will try to reduce gender disparity without using sex as a feature by reweighting the training samples to give higher weight to female records during training. Hypothesis: upweighting females will increase their predicted scores and narrow the positive-rate gap and AUC gap between sexes. I may also try dropping occupation (a potential proxy) if needed.
- Thought: I will explore additional proxy feature removal to improve gender neutrality without using the sex feature. Hypothesis: dropping occupation (and possibly workclass and race) will reduce gender proxy signals in the splits, narrowing the male/female disparities in AUC and positive rate, at some cost to overall AUC. I will try several drop sets (keeping numeric features, education, native-country).
- Thought: Aim: reduce gender proxy effects further by removing numeric features that may indirectly encode gender patterns, especially hours-per-week (known to differ by sex). Hypothesis: dropping hours-per-week (and possibly fnlwgt) will narrow positive-rate gaps and AUC gaps by sex, at some cost to overall AUC.
- Thought: I will target fairness by increasing predicted probabilities at leaves that predominantly contain female samples, without using sex as an input feature. Hypothesis: modestly increasing leaf values for female-heavy leaves will raise female positive rates more than male (since more females route to those leaves), narrowing the gap while keeping ROC AUC nearly unchanged. Plan: - Identify leaves and compute, for each leaf, the val-sample counts and female share. - For candidate uplift deltas (0.02, 0.05, 0.08), adjust leaf values where female share >= 0.6 (and also try >= 0.55) with clipping to [0,1]. - Evaluate val AUC and fairness metrics; choose the best tree by fairness score, tie-breaking by val AUC. - Keep everything reproducible and print diagnostics.