SYNTHONY: A STRESS-AWARE, INTENT-CONDITIONED AGENT FOR DEEP TABULAR GENERATIVE MODELS SELECTION

Hochan Son*
Department of Statistics
University of California, Los Angeles
hochanson@g.ucla.edu

Xiaofeng Lin*
Department of Statistics
University of California, Los Angeles
bernardo1998@ucla.edu

Jason Ni
Department of Mathematics
University of California, Los Angeles
jasonni19@g.ucla.edu

Guang Cheng
Department of Statistics
University of California, Los Angeles
guangcheng@stat.ucla.edu

ABSTRACT

Deep generative models for tabular data (GANs, diffusion models, and LLM-based generators) exhibit highly non-uniform behavior across datasets; the best-performing synthesizer family depends strongly on distributional stressors such as long-tailed marginals, high-cardinality categoricals, Zipfian imbalance, and small-sample regimes. This brittleness makes practical deployment challenging, especially when users must balance competing objectives of fidelity, privacy, and utility.

We study *intent-conditioned tabular synthesis selection*: given a dataset and a user intent expressed as a preference over evaluation metrics, the goal is to select a synthesizer that minimizes regret relative to an intent-specific oracle. We propose **stress profiling**, a synthesis-specific meta-feature representation that quantifies dataset difficulty along four interpretable stress dimensions, and integrate it into **SYNTHONY**, a selection framework that matches stress profiles against a calibrated capability registry of synthesizer families. Across a benchmark of 7 datasets, 10 synthesizers, and 3 intents, we demonstrate that stress-based meta-features are highly predictive of synthesizer performance: a kNN selector using these features achieves strong Top-1 selection accuracy, substantially outperforming zero-shot LLM selectors and random baselines. We analyze the gap between meta-feature-based and capability-based selection, identifying the hand-crafted capability registry as the primary bottleneck and motivating learned capability representations as a direction for future work.

1 INTRODUCTION

The central challenge in tabular data synthesis is no longer *access* to generative models, but the *selection* of them. GANs, diffusion models, and LLM-based generators now offer diverse architectures for producing synthetic tabular data (Xu et al., 2019; Kotelnikov et al., 2023; Borisov et al., 2022), addressing data scarcity, privacy constraints, and class imbalance (Bauer et al., 2024; Chawla et al., 2002). Yet unlike supervised tabular learning, where AutoML frameworks automate the search from preprocessing to hyperparameter tuning (Feurer et al., 2015; Erickson et al., 2020), the synthesis landscape lacks a comparable "Auto-Synthesizer" paradigm. Practitioners must manually select between architectures ranging from copulas to diffusion models, a choice that is particularly risky when an untuned complex model (e.g., a GAN) underperforms a simple, well-tuned baseline due to training instability or mode collapse (Mescheder et al., 2018).

We identify four canonical failure modes that expose the limitations of uninformed model selection:

*Equal contribution.

- **The Long Tail Problem**: Datasets with severe distributional skew cause mode collapse in basic GANs and VAEs, which optimize for central tendencies rather than tail fidelity.

- **The Zipfian Imbalance Problem**: Categorical columns following power-law distributions (Newman, 2005) cause generative models with continuous latent bottlenecks to silently erase rare-but-critical tail categories—a failure invisible to aggregate fidelity metrics yet catastrophic for minority-sensitive downstream tasks.

- **The Needle in the Haystack Problem**: High-cardinality categorical variables (hundreds or thousands of unique values) overwhelm embedding-based encoders, leading to mode collapse even when the underlying distribution is not heavily skewed.

- **The Small Data Trap**: With limited training samples, expressive deep models memorize rather than generalize, producing synthetic records that leak private information (Stadler et al., 2022).

These failure modes are not random – they are predictable from measurable dataset properties, which motivates a profiling-based approach to selection.

Current open-source libraries, such as SDV (Patki et al., 2016) and SynthCity (Qian et al., 2023), have made significant strides in democratizing access to individual synthesizers. However, they function as collections of independent building blocks rather than intelligent selection systems. Users must still manually navigate the trade-offs between model expressiveness, computational cost, and data compatibility, a task requiring deep expertise in both generative modeling and dataset characteristics.

To bridge this gap, we study *intent-conditioned synthesis selection* for deep tabular generative modeling. The core hypothesis is that tabular DGMs fail in systematic, predictable ways that depend on dataset stressors (e.g., long tails, high-cardinality categoricals, Zipfian imbalance, and small-sample regimes). We operationalize this hypothesis via **stress profiling**: a synthesis-specific meta-feature representation that quantifies dataset difficulty along four interpretable dimensions and maps them to known failure modes. We then build **SYNTHONY**, a selection framework that matches stress profiles against a calibrated capability registry to rank synthesizer families for a given dataset and intent.

Our contributions are:

- **Problem formulation and evaluation.** We formalize *intent-conditioned tabular synthesis selection* and evaluate selectors using Top-K accuracy, Spearman rank correlation, and NDCG against intent-specific oracles.

- **Stress profiling as a predictive principle.** We introduce *stress profiles*, four synthesis-specific, interpretable dataset stress dimensions, and demonstrate empirically that they predict which deep generative model families succeed or fail on tabular data.

- **Capability-based selection framework.** We propose SYNTHONY, which uses stress profiles to score candidate model families via a capability registry grounded in architectural properties and synthesis literature. The registry encodes domain knowledge (e.g., "diffusion models handle skew better than basic VAEs") as structured, interpretable scores rather than opaque learned weights, enabling practitioners to inspect and override individual entries. Intent-conditioned scale factors are calibrated from data via Bayesian optimization. We further analyze the gap between meta-feature-based and capability-based selection, identifying directions for learned capability representations.

- **Cross-family empirical analysis.** We benchmark 10 synthesizers across 4 families, 7 datasets, and 3 user intents, revealing that optimal model selection is strongly intent-dependent and that stress-based meta-features capture this dependence more reliably than heuristic rules or zero-shot LLM reasoning.

2 Related Work

2.1 Generative Models for Tabular Data

The landscape of tabular synthesis has evolved from statistical methods (Bayesian Networks (Zhang et al., 2014), Gaussian Copulas (Patki et al., 2016)) through GANs adapted for mixed-type data such as CTGAN (Xu et al., 2019) and TableGAN (Mahmood, 2023), to diffusion models that now represent the state-of-the-art: TabDDPM (Kotelnikov et al., 2023) introduced cascading Gaussian and multinomial diffusion for mixed types; TabSyn (Zhang et al., 2024) performs diffusion in a VAE-crafted latent space with significant gains in quality and speed. Concurrently, LLMs have been adapted for tabular synthesis (GReaT, REaLTabFormer (Borisov et al., 2022; Solatorio & Dupriez, 2023)). This architectural diversity creates a selection challenge: without systematic guidance, choosing the optimal generator for a given dataset and constraints remains trial-and-error.

2.2 Benchmarking and Evaluation

Existing synthesis libraries focus on *standardization* and *evaluation* rather than automated *selection*. SDV (Patki et al., 2016) and SynthCity (Qian et al., 2023) provide unified APIs and benchmarking but require manual hyperparameter specification. SynMeter (Du & Li, 2024) and STNG (Rashidi et al., 2024) integrate tuning loops into evaluation, yet still leave model *selection* to the user. Our work introduces an automated selection layer: unlike these tools, SYNTHONY automates the upstream decision of *which* model to use by matching dataset stress profiles against a calibrated capability registry conditioned on user intent.

2.3 Meta-Learning and Algorithm Selection

Algorithm selection—the problem of choosing the best solver based on instance features— was formalized by Rice (1976) and is mature for supervised learning (Vanschoren, 2019), where dataset meta-features enable warm-started search in Auto-sklearn (Feurer et al., 2015) and AutoGluon (Erickson et al., 2020) without exhaustive evaluation (Rivolli et al., 2022). However, this paradigm has not been applied to synthesis. Recent surveys consistently identify synthesizer selection as an open problem: Kindji et al. (2025) find rankings shift with compute budget but offer no predictive mechanism; Mayer et al. (2025) conclude practitioners must manually interpret results; and Davila et al. (2024) construct a qualitative guide but stop short of automation. Our work addresses this gap with **stress profiling**, a synthesis-specific meta-feature framework quantifying dataset difficulty along four dimensions corresponding to known generative failure modes, and SYNTHONY, which uses these profiles for predictive, intent-conditioned selection, complementing existing libraries (Patki et al., 2016; Qian et al., 2023) by automating the question of *which* synthesizer to deploy.

3 Methodology

3.1 Problem Formulation

We define the tabular synthesis selection problem as a tuple $\mathcal{T} = (\mathcal{D}, \mathcal{I}, \mathcal{G})$, where $\mathcal{D}$ is the private source dataset, $\mathcal{G} = \{g_1, \ldots, g_k\}$ is a set of candidate generative architectures (e.g., GANs, diffusion models, copulas), and $\mathcal{I}$ represents the user's intent. The intent $\mathcal{I}$ specifies a preference over competing evaluation dimensions such as fidelity, privacy, and utility.

Our objective is to identify the optimal generator g^* such that:

$$g^* = \arg\max_{g \in \mathcal{G}} \ \mathcal{F}(\mathbf{m}_g \mid \mathcal{I}) \tag{1}$$

where $\mathbf{m}_g$ is the evaluation metric vector for generator g on dataset $\mathcal{D}$, and $\mathcal{F}$ is a scalarization function derived from the user's intent $\mathcal{I}$.

Equation 1 defines the *oracle* selection, which requires evaluating every generator. Since this is prohibitively expensive, SYNTHONY approximates it via a surrogate scoring function (Equation 2) that operates on pre-computed capability scores rather than evaluation metrics.

3.2 STRESS PROFILING: DATASET DIFFICULTY SIGNATURES

Given a dataset D with mixed continuous and categorical features, we compute a stress profile $\phi(D) \in \mathbb{R}^4$ capturing four synthesis-relevant stressors: (i) **Long-tail skew**: maximum absolute Fisher–Pearson skewness across numeric columns (flagged if any column exceeds $|\text{skew}| > 2.0$); (ii) **Cardinality complexity**: maximum unique-value count across all columns (flagged if any column exceeds 500 unique values); (iii) **Zipfian concentration**: maximum top-20% frequency ratio across categorical columns, i.e., the fraction of rows occupied by the most frequent 20% of categories (flagged as high-stress if > 0.80); and (iv) **Small-sample regime**: total row count (flagged if < 500). These stressors correspond directly to the failure modes identified in Section 1: long-tail skew maps to the Long Tail Problem, the Needle in Haystack Problem maps jointly to cardinality complexity and Zipf concentration, and the Small Data Trap maps to the small-sample regime.

Stress profiles serve two roles: (1) they enable interpretable diagnosis of why a synthesizer fails on a given dataset, and (2) they provide meta-features for predicting promising model families under a fixed evaluation budget.

From stress profiles to requirement vectors. The 4-dimensional stress profile $\phi(D) \in \mathbb{R}^4$ is mapped to a 6-dimensional requirement vector $\mathbf{r} \in \{0, \ldots, 4\}^6$ used for capability scoring. Each raw stress statistic is quantized into a discrete requirement level via threshold-based binning. For example, long-tail skew maps to `skew_handling` as: $r = 0$ if $|\text{skew}| < 0.5$ (no stress), $r = 1$ if < 1.0, $r = 2$ if < 2.0, $r = 3$ if < 3.0, and $r = 4$ if ≥ 3.0. Cardinality complexity maps to `cardinality_handling` via unique-value-count bins ($< 50 \to 0$, $< 200 \to 1$, $< 500 \to 2$, $< 1000 \to 3$, $\geq 1000 \to 4$). Zipf concentration maps to `zipfian_handling` via top-20% frequency ratio bins ($< 0.5 \to 0$, $< 0.65 \to 1$, $< 0.80 \to 2$, $< 0.90 \to 3$, $\geq 0.90 \to 4$). Small-sample regime maps to `small_data` via row count ($\geq$ 10,000 $\to 0$, $\geq$ 5,000 $\to 1$, $\geq$ 1,000 $\to 2$, $\geq 500 \to 3$, $< 500 \to 4$). A fifth dimension, `correlation_handling`, is derived from a separate correlation detector (which counts the fraction of feature pairs with absolute Pearson correlation exceeding 0.7) that quantifies inter-feature dependency density. The sixth dimension, `privacy_dp`, does not enter the requirement vector directly; instead, it influences scoring only through intent-conditioned scale factors (Section 3.4), allowing the privacy intent to upweight differential privacy capabilities without imposing them as a hard requirement for all datasets.

3.3 STRESS-AWARE CAPABILITY SCORING

SYNTHONY uses stress profiles to score and rank candidate model families *without* running each generator. For each generator $g \in \mathcal{G}$, we maintain a capability registry $\mathbf{C}_g \in \{0, \ldots, 4\}^J$ scoring the model's expected robustness along J capability dimensions (e.g., handling of skewed distributions, high cardinality, small samples). Given a stress profile $\phi(D)$, SYNTHONY derives a requirement vector $\mathbf{r}(\phi(D))$ indicating which capabilities the dataset demands.

To account for the relative importance of capabilities under different intents, we introduce intent-conditioned scale factors $\boldsymbol{\alpha}_i = (\alpha_{i,1}, \ldots, \alpha_{i,J}) \in [0, 10]^J$ for each intent $i \in \mathcal{I}$. The final score for generator g under intent i is:

$$\text{Score}(g, i) = \sum_{j=1}^{J} \alpha_{i,j} \cdot c_{g,j} \cdot r_j(\phi(D)) \tag{2}$$

where $c_{g,j}$ is the capability score and $r_j(\phi(D))$ is the stress-derived requirement for dimension j. Generators are ranked by this score, and the top-ranked model is recommended.

The match function underlying Equation 2 implements soft thresholding with graceful degradation: full credit (1.0) when the model's capability meets or exceeds the requirement, partial credit (0.7, 0.4) for near-misses, and zero otherwise. The full specification of the requirement derivation f and match function m_j is given in Appendix B.

Additive independence assumption. The scoring function in Equation 2 treats capability dimensions independently: each dimension contributes additively, so a model that is weak in one area is not penalized in another. This design enables interpretable, per-dimension scoring but may miss interaction effects—for instance, a dataset with *both* high cardinality and severe skew may pose

Algorithm 1 SYNTHONY Stress-Aware Selection

Require: Dataset $\mathcal{D}$, Intent $\mathcal{I}$, Calibrated registry $\mathbf{C}$, Scale factors $\boldsymbol{\alpha}$
Ensure: Ranked generator list
1: Compute stress profile $\phi(\mathcal{D})$
2: Derive requirement vector $\mathbf{r} \leftarrow f(\phi(\mathcal{D}))$
3: **for** each generator $g \in \mathcal{G}$ **do**
4: Score$(g) \leftarrow \sum_j \alpha_{\mathcal{I},j} \cdot c_{g,j} \cdot r_j$
5: **end for**
6: **return** Generators ranked by Score(g) descending

compounding challenges that exceed the sum of individual stresses. Capturing such interactions would require either higher-order terms in the scoring function or a learned model that implicitly discovers them from data.

3.4 Calibration via Bayesian Optimization

Rather than relying solely on hand-crafted capability scores, we calibrate the entire scoring system against empirical benchmark data. We jointly optimize two parameter groups using Bayesian optimization: (1) the capability scores $c_{g,j} \in \{0, \ldots, 4\}$ for each generator g and capability dimension j, and (2) the intent-conditioned scale factors $\alpha_{i,j} \in [0, 10]$ that weight each capability under intent i. For $|\mathcal{G}| = 10$ generators, $|J| = 6$ capabilities, and $|\mathcal{I}| = 3$ intents, this yields $10 \times 6 + 3 \times 6 = 78$ parameters. We maximize the average Spearman rank correlation between predicted and oracle rankings on the training split using Tree-structured Parzen Estimator (TPE) sampling (Bergstra et al., 2011) (500 trials, 50 startup). Spearman correlation is chosen over Top-1 accuracy because it rewards correct ordering across the full ranking, not just the top pick.

Algorithm 1 summarizes the full SYNTHONY selection pipeline.

4 Experiments

To validate the effectiveness of SYNTHONY as a model selection agent, we conduct a systematic evaluation comparing its choices against intent-conditioned ground truth. Our goal is to answer: *Can a stress-aware, intent-conditioned agent identify the optimal synthesizer for a given dataset and objective more reliably than heuristic rules, zero-shot LLMs, or meta-learning selectors?*

4.1 Experimental Setup

Benchmark Datasets We use 7 tabular datasets sourced from OpenML (Vanschoren et al., 2013) spanning diverse characteristics: Abalone, Bean, IndianLiverPatient, Obesity, faults, insurance, and wilt. These datasets vary in dimensionality (6–17 features), sample size (579–13,611 rows), and feature types (continuous, categorical, mixed), creating a robust test bed. We evaluate under three intents (privacy, fidelity, utility), yielding $7 \times 3 = 21$ dataset-intent pairs. We split at the dataset level (seed=42): 4 datasets for training and 3 for testing, producing $4 \times 3 = 12$ training pairs and $3 \times 3 = 9$ test pairs. Dataset-level splitting ensures that no dataset appears in both splits, preventing trivial memorization. All tables report held-out test set performance unless noted otherwise.

Candidate Generators For each dataset, we evaluate a pool of N=10 synthesizers spanning four categories: **Differential Privacy** (AIM, DPCART), **Deep Generative** (AutoDiff, TabDDPM, TVAE, NFlow), **Tree-Based** (ARF, CART), and **Statistical** (BayesianNetwork, SMOTE). We exclude the Identity (copy) baseline, as it trivially achieves perfect fidelity but zero privacy, making it uninformative for selection. One dataset-model pair (insurance, SMOTE) is absent, so insurance rankings contain 9 models while all others contain 10.

Ground Truth (Intent-Conditioned Oracles) To establish ground truth, we train every generator $g \in \mathcal{G}$ on every dataset $d \in \mathcal{D}_{\text{bench}}$ and compute intent-specific metrics: **Privacy** (Proportion Closer to Real, lower is better; for ranking we invert so that higher rank = better privacy), **Fidelity**

(Column Shape Score, higher is better), and **Utility** (downstream ML task performance: test ROC AUC for classification datasets, test adjusted R^2 for regression datasets; higher is better). Privacy and fidelity metrics are sourced from the benchmark's cleaned results. Utility scores are computed from a separate Tabular Utility evaluation, which trains classifiers/regressors on synthetic data and evaluates on held-out real test data. For each dataset-intent pair, we rank all available models by the corresponding metric, producing an oracle ranking $g_d^*(\mathbf{w})$.

4.2 BASELINES

We compare SYNTHONY against four selection strategies. **Static Heuristic:** a logic-based baseline mimicking practitioner heuristics using only row count and intent (tree-based for utility/fidelity, DP models for privacy), without dataset profiling beyond row count. **Vanilla LLM Selector:** a GPT-4o-mini zero-shot selector given the dataset schema and summary statistics but no stress profiles or capability scores (temperature 0). **Random Search:** uniform random generator sampling (expected value over 1,000 trials). **Meta-feature kNN Selector:** a meta-learning baseline extracting 9 dataset meta-features, finding k=3 nearest training datasets by Euclidean distance, and aggregating their oracle rankings. Because it interpolates from oracle rankings of training datasets, kNN has a structural advantage over zero-knowledge methods: it leverages ground-truth evaluation data at training time. We report it above the midrule in Table 1 as an *oracle-informed reference point* rather than a deployable zero-knowledge method.

4.3 EVALUATION METRICS

We assess selection quality with four metrics computed over the 10-model rankings. Our two primary metrics are **Top-3 Accuracy**, the fraction of dataset-intent pairs where the oracle-best model appears in the agent's top-3 recommendations (reflecting the realistic deployment scenario of presenting a short list for downstream evaluation), and **Spearman Rank Correlation**, which measures agreement between predicted and oracle full rankings across all 10 models ($[-1, 1]$) and rewards correct ordering across the entire ranking. As secondary metrics we report **Top-1 Accuracy**, the fraction of pairs where the selector's top pick matches the oracle-best model (with only 9 test pairs, each correct prediction shifts Top-1 by 11.1 percentage points, so we interpret differences smaller than $\sim$0.15 with caution), and **NDCG** (Normalized Discounted Cumulative Gain), which evaluates the full predicted ranking with position-weighted relevance ($[0, 1]$).

4.4 SYNTHONY OPTIMIZATION VARIANTS

We evaluate two optimization strategies. **SF-only** tunes 18 scale factor parameters (6 capabilities $\times$ 3 intents) via Bayesian optimization (Optuna, 200 trials, TPE sampler), holding hand-crafted capability scores fixed, with Top-1 accuracy as the training objective. **Joint optimization** simultaneously tunes 78 parameters (60 capability scores, $|\mathcal{G}|$=10 $\times$ $|J|$=6, integer $\{0, \ldots, 4\}$; plus 18 scale factors, float $[0, 10]$) over 500 trials, optimizing average Spearman on the training set. We use Spearman for the larger search space because Top-1 is too sparse a signal for 78 parameters. These variants differ in both parameter count and objective, so performance differences reflect both changes; controlled variants appear in Appendix A.

4.5 RESULTS

Selection Performance Tabular synthesizers exhibit highly non-uniform performance across datasets and intents. Privacy exhibits genuine dataset-dependent variation: the best model varies across datasets, with multiple model families each winning on different datasets. Fidelity and utility also show dataset-dependent patterns, though with somewhat stronger concentration among tree-based and statistical models. This motivates intent-conditioned selection: no single model can serve all objectives.

Table 1 summarizes test set performance across 9 held-out dataset-intent pairs. We highlight Top-3 accuracy and Spearman correlation as the primary comparison metrics.

Among zero-knowledge methods, SYNTHONY (SF-only) achieves the best Spearman ($\rho = 0.573$) and Top-1 (0.444), substantially outperforming the Static Heuristic (0.422 Spearman) and the Vanilla

Table 1: Test set performance of intent-conditioned model selection strategies (9 dataset-intent pairs). Primary metrics are *italicized*. kNN† uses oracle neighbor rankings from training datasets and serves as an *oracle-informed reference point*, not a deployable zero-knowledge method. Bold indicates best among zero-knowledge methods. With 9 test pairs, each Top-1 prediction shifts accuracy by $\approx$11.1 percentage points.

Strategy	Top-1	*Top-3*	*Spear.*	NDCG
kNN (k=3)† *(oracle-informed)*	.556	*.778*	*.578*	.918
Random (E[1000])	.111	*.304*	$-.001$	.815
Vanilla LLM (GPT-4o-mini)	.000	*.111*	$-.087$	.784
Static Heuristic	.333	*.778*	*.422*	.889
SYNTHONY (joint opt)	.000	*.556*	*.557*	.893
SYNTHONY (SF-only)	**.444**	***.667***	***.573***	**.914**

LLM (-0.087, worse than random). The SF-only variant nearly matches the oracle-informed kNN reference (0.578 Spearman) despite a fundamental asymmetry: kNN interpolates from ground-truth rankings of training datasets, while SYNTHONY requires *no* oracle scores at inference time. This makes SYNTHONY applicable to new synthesizers added to the registry without rerunning benchmarks—a practical advantage kNN cannot offer. Furthermore, SYNTHONY's capability-based scoring provides interpretable, per-dimension explanations for each recommendation, whereas kNN's neighbor-aggregation is opaque about *why* a model is suitable.

The joint optimization does not generalize as well: test Spearman (0.557) is slightly below SF-only, and Top-1 drops to 0.000. This suggests overfitting when optimizing 78 parameters on 12 training pairs, though the comparison is confounded by different training objectives (Top-1 vs. Spearman). The Static Heuristic achieves high Top-3 (77.8%) by placing strong models within the top 3, but its poor Spearman (0.422) indicates weak full-ranking quality.

Per-Intent Breakdown To disentangle selection quality from CART's frequent wins (8/21 oracle-best across all pairs), we examine per-intent oracle patterns across all 7 benchmark datasets, not just 3 test dataset. Under **fidelity**, CART wins 4/7 datasets; under **utility**, CART wins 4/7 while SMOTE takes the remaining 3. The discriminative test is **privacy**, where five distinct models win across seven datasets (TabDDPM 2, AutoDiff 2, AIM/DPCART/BayesianNetwork each 1). A selector that simply always predicts CART would achieve $8/21 = 38\%$ Top-1 overall but 0% on privacy. SYNTHONY's stress-aware scoring provides genuine signal beyond predicting the majority class: it differentiates utility from fidelity requirements and identifies non-CART winners on datasets where tree-based models underperform.

On the 9 test pairs specifically, SYNTHONY (SF-only) correctly identifies the oracle-best model on 4/9 pairs (Top-1 = 0.444). Its errors concentrate on the privacy intent, the hardest for all methods—consistent with the diversity of oracle winners under privacy across the full benchmark. The Static Heuristic achieves 3/9 Top-1, but all its correct predictions fall under fidelity and utility where CART dominates; it achieves 0% Top-1 on privacy. SYNTHONY's correct privacy predictions, though limited, demonstrate that stress profiling captures signals that simple heuristics miss.

CART Dominance and Benchmark Scope CART's strong showing (8/21 oracle-best overall) partly reflects the composition of our 7-dataset benchmark: most datasets are moderate-sized (500–$13{,}000$ rows) with relatively low cardinality, a regime where tree-based methods are known to excel. We note this is consistent with recent independent benchmarks (Mayer et al., 2025; Kindji et al., 2025) that also find tree-based methods competitive in this data range. Expanding to datasets with higher cardinality, more features, or larger scale ($>100{,}000$ rows) would likely shift the distribution of oracle winners toward deep generative models and reduce single-model dominance.

Privacy Intent: Challenges and Signal The privacy intent presents a circular challenge: it is where the discriminative signal for model selection is strongest (five distinct oracle winners across seven datasets), but it is also the hardest to model because no single architectural property reliably predicts privacy performance. DP-certified models (AIM, DPCART) do not always achieve the best

Table 2: Ablation study on the SF-only pipeline (test set, 9 pairs). Components: (A) stress profiling, (B) capability matching, (C) focus scaling. Stress profiling provides the strongest signal; adding focus scaling degrades test performance due to overfitting at this sample size (n=12 training pairs, 18 parameters).

Variant	Top-1	Top-3	Spear.	NDCG
Vanilla LLM (no SYNTHONY)	.000	.111	$-.087$	.784
− Stress − Focus (bare scoring)	.444	.667	.478	.904
− Focus scaling (stress only)	.444	.667	.602	.910
− Stress profiling (focus only)	.333	.667	.430	.901
Full SYNTHONY (SF-only)	.444	.667	.573	.914

Table 3: Predicting the best synthesizer from dataset meta-features (test set, 9 pairs). Majority vote's 33.3% Top-1 reflects CART's frequent wins (8/21 oracle-best); the discriminative signal is in Spearman, where kNN (0.578) substantially outperforms frequency-based and parametric baselines (0.293–0.300).

Predictor	Top-1	Top-3	Spear.	NDCG
Majority vote	.333	.778	.300	.899
Logistic regression	.556	.889	.293	.893
Decision tree	.222	.667	.299	.900
kNN (k=3)	.556	.778	.578	.918

privacy scores, as non-DP models like TabDDPM can achieve lower re-identification risk through smoother density estimation. This makes privacy the most compelling use case for stress-aware selection—and the one requiring the most training data to calibrate reliably.

4.6 ABLATION STUDY

To isolate the contribution of each SYNTHONY component, we systematically disable them (Table 2). SYNTHONY's SF-only pipeline consists of three components: (A) stress profiling (detecting required capabilities from dataset statistics), (B) capability matching (scoring models against requirements using the capability registry), and (C) focus-based scaling (Bayesian-optimized scale factors per intent). All ablation variants use the SF-only optimization (18 parameters) to isolate component-level effects.

Table 2 reveals an interesting pattern. Removing focus scaling ("− Focus scaling") actually *improves* Spearman from 0.573 to 0.602, suggesting that the optimized scale factors slightly overfit on 12 training pairs. The stress-only variant achieves the best Spearman among all ablations while maintaining Top-1 (0.444) and Top-3 (0.667) accuracy. This does not invalidate intent-based scaling as a concept, but indicates that 12 training pairs are insufficient to learn 18 scale parameters reliably. We conjecture that with a larger benchmark ($\geq$20 datasets), focus scaling would become beneficial—particularly for the privacy intent, where intent-specific weighting should upweight DP capability for privacy-focused users.

Removing stress profiling ("− Stress profiling") causes the largest Spearman drop ($0.573 \rightarrow 0.430$), confirming that dataset-derived capability requirements are the most important signal. Without stress profiling, the system cannot differentiate which models suit which datasets; the remaining score differences come only from intent-conditioned scale factors applied to raw capability scores.

The bare scoring variant ("− Stress − Focus") still achieves 0.478 Spearman, showing that even unweighted capability matching provides signal above random (-0.001). The full system combines both components to achieve the best overall balance across all four metrics.

4.7 Do Stress Profiles Predict Synthesizer Families?

Table 3 validates that dataset meta-features (including stress profile dimensions) are predictive of which synthesizer succeeds. Logistic regression achieves the highest Top-1 (0.556) and Top-3 (0.889) accuracy, but kNN (k=3) achieves the best Spearman correlation (0.578) by nearly double, demonstrating that meta-feature similarity produces better *full rankings*, not just correct top-1 predictions. This supports the stress profiling hypothesis: dataset difficulty signatures predict synthesizer efficacy in a systematic, learnable way.

Implications for SYNTHONY. Remarkably, SYNTHONY (SF-only) nearly matches the kNN stress predictor on Spearman (0.573 vs. 0.578) despite using a fundamentally different approach: capability matching rather than neighbor-based interpolation. The joint optimization does not improve on this, suggesting that the hand-crafted capability scores already encode useful inductive bias that is lost when all 78 parameters are freely optimized. The remaining gap with kNN reflects a structural difference: kNN directly interpolates oracle rankings from similar datasets, while SYNTHONY must generalize through a capability-based abstraction layer. Crucially, this near-parity is achieved without access to oracle rankings at test time—SYNTHONY's capability registry generalizes to new datasets without rerunning any synthesizer, while kNN requires precomputed rankings for all training datasets.

Analysis of "Hard" Datasets The privacy intent is the most challenging: no single model dominates, and the best choice varies across datasets (TabDDPM wins 2/7, AutoDiff wins 2/7, AIM/DP-CART/BayesianNetwork each win 1/7). All methods exhibit the lowest accuracy on privacy, where the Static Heuristic assumption that DP-certified models (DPCART, AIM) always minimize re-identification risk is violated: on datasets with moderate dimensionality, non-DP diffusion models (TabDDPM or AutoDiff) achieve lower Distance-to-Closest-Record scores by learning smoother density estimates that avoid near-copying. The kNN selector handles privacy best by aggregating neighbor-specific patterns rather than relying on a fixed rule. Utility presents an interesting challenge: CART wins 4/7 datasets but SMOTE takes the other 3, and the best synthesizer depends on both dataset characteristics and the downstream task type (classification vs. regression), requiring selectors to account for this interaction.

5 Conclusion

We studied intent-conditioned model selection for tabular data synthesis and proposed **stress profiling**, a set of interpretable meta-features capturing dataset difficulty, integrated into SYNTHONY, a selection framework that matches stress profiles against a calibrated capability registry.

Our evaluation yields three main findings. First, **stress profiling is predictive**: a kNN meta-learner achieves 77.8% Top-3 and 55.6% Top-1 on held-out pairs (Table 3), confirming that DGM failures are systematic and predictable from dataset characteristics. Second, **capability matching nearly reaches oracle-informed meta-learning**: SYNTHONY (SF-only) achieves 0.573 Spearman, nearly matching kNN (0.578), while offering practical advantages that kNN lacks—zero-shot applicability to new synthesizers (no benchmark rerunning needed), interpretable per-dimension explanations for each recommendation, and the ability to incorporate domain knowledge through the capability registry. The joint optimization (0.557 Spearman) does not improve on the test set, indicating that the hand-crafted capability scores encode useful inductive bias lost when all 78 parameters are freely optimized on 12 training pairs. Third, **zero-shot LLMs are insufficient**: GPT-4o-mini achieves 0% Top-1 and -0.087 Spearman, confirming that synthesis-specific calibration is essential.

Limitations. Our 7-dataset, 9-pair test set limits statistical power. CART wins 4/7 fidelity and 4/7 utility pairs, making the problem partially solvable by majority-class prediction (33.3% Top-1). The SF-only and joint variants differ in both parameter space and objective (Top-1 vs. Spearman), partially confounding the comparison; controlled variants appear in Appendix A.

Future Work. The primary direction is *learning* capability scores from data rather than hand-crafting them: replacing the static registry with learned embeddings (Vanschoren, 2019), incorporating stress meta-features directly into the scoring function, and expanding to $\geq$20 datasets and

additional model families (e.g., GReaT (Borisov et al., 2022)) to reduce single-model dominance effects.

Reproducibility Statement

Code, data, and the capability registry are publicly available at `https://github.com/UCLA-Trustworthy-AI-Lab/Synthony`.

References

André Bauer, Simon Trapp, Michael Stenger, Robert Leppich, Samuel Kounev, Mark Leznik, Kyle Chard, and Ian Foster. Comprehensive exploration of synthetic data generation: A survey. *arXiv preprint arXiv:2401.02524*, 2024.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

Vadim Borisov, Kathrin Seßler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. Language models are realistic tabular data generators. *arXiv preprint arXiv:2210.06280*, 2022.

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002. doi: 10.1613/jair.953.

Avrami Davila, Akim Shin, Shreya Kamthe, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Navigating tabular data synthesis research: Understanding user needs and tool capabilities. *SIGMOD Record*, 53(2):51–64, 2024. doi: 10.1145/3685980.3685992.

Yuntao Du and Ninghui Li. Systematic assessment of tabular data synthesis algorithms. *arXiv preprint arXiv:2402.06806*, 2024.

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

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Komivi Dodzi Kindji, Bernd Malle, Peter Kieseberg, and Andreas Holzinger. Tabular data generation models: An in-depth survey and performance benchmarks with extensive tuning. *Neurocomputing*, 617:128984, 2025. doi: 10.1016/j.neucom.2024.128984.

Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. TabDDPM: Modelling tabular data with diffusion models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 17564–17579. PMLR, 2023.

Mahmood Mahmood. tablegan: A gan-based approach for generating synthetic tabular data. `https://github.com/mahmoodm2/tableGAN`, 2023.

Rudolf Mayer, Tatjana Welzer, and Mojca Indihar Štemberger. Benchmarking tabular data synthesis: Evaluating tools, metrics, and datasets on prosumer hardware for end-users. *Data Science Journal*, 24(1):37, 2025. doi: 10.5334/dsj-2025-037.

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3481–3490. PMLR, 2018.

Mark EJ Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5): 323–351, 2005.

Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 399–410, 2016. doi: 10.1109/DSAA.2016.49.

Michael Platzer and Thomas Reutterer. Holdout-based empirical assessment of mixed-type synthetic data. *Frontiers in big Data*, 4:679939, 2021.

Zhaozhi Qian, Bogdan-Constantin Cebere, and Mihaela van der Schaar. Synthcity: facilitating innovative use cases of synthetic data in different data modalities. *arXiv preprint arXiv:2301.07573*, 2023.

Hooman H. Rashidi, Samer Albahra, Brian P. Rubin, and Bo Hu. A novel and fully automated platform for synthetic tabular data generation and validation. *Scientific Reports*, 14:23312, 2024. doi: 10.1038/s41598-024-73608-0.

John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Adriano Rivolli, Luís PF Garcia, Carlos Soares, Joaquin Vanschoren, and André CPLF de Carvalho. Meta-features for meta-learning. *Knowledge-Based Systems*, 240:108101, 2022.

Aivin V. Solatorio and Olivier Dupriez. REaLTabFormer: Generating realistic relational and tabular data using transformers. *arXiv preprint arXiv:2302.02041*, 2023.

Theresa Stadler, Bristena Oprisanu, and Carmela Troncoso. Synthetic data – anonymisation groundhog day. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 1451–1468. USENIX Association, 2022.

Joaquin Vanschoren. Meta-learning. In *Automated Machine Learning*, pp. 35–61. Springer, 2019.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198.

Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional GAN. *arXiv preprint arXiv:1907.00503*, 2019.

Hengrui Zhang, Jiani Zhang, Balasubramaniam Srinivasan, Zhengyuan Shen, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. In *International Conference on Learning Representations*, 2024.

Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private data release via bayesian networks. In *Proceedings of the 32nd ACM SIGMOD International Conference on Management of Data*, pp. 1423–1434. ACM, 2014. doi: 10.1145/2588555.2588573.

A JOINT OPTIMIZATION OF CAPABILITY SCORES AND SCALE FACTORS

This appendix provides full details of the Bayesian optimization procedure used to calibrate SYNTHONY's scoring system against empirical benchmark data.

A.1 MOTIVATION

SYNTHONY's recommendation engine scores candidate synthesizers by matching dataset stress profiles against a capability registry. Each model g has a capability vector $\mathbf{c}_g \in \{0, \ldots, 4\}^6$ across six dimensions: skew handling, cardinality handling, Zipfian handling, small data, correlation handling, and differential privacy. Under a given intent i, scale factors $\boldsymbol{\alpha}_i \in [0, 10]^6$ weight each capability dimension, so the total score for model g on dataset d under intent i is:

$$S(g, d, i) = \sum_{j=1}^{6} \alpha_{i,j} \cdot w_j(d) \cdot m_j(c_{g,j}, r_j(d)) \tag{3}$$

where $r_j(d)$ is the required capability level derived from the dataset stress profile, $w_j(d)$ is a base weight (1.0 if capability j is required, 0.1 otherwise), and m_j is a match function that compares $c_{g,j}$ against $r_j(d)$.

The initial capability scores were assigned by human judgment based on model architecture properties and the literature. Before calibration (i.e., with uniform scale factors), the SF-only variant with hand-crafted capabilities achieves 0.444 test Top-1 and 0.573 Spearman (Table 1), showing that the hand-crafted scores already encode useful inductive bias but leave room for improvement through calibration.

A.2 PARAMETER SPACE

We jointly optimize two parameter groups:

Capability scores (60 parameters). For each of the 10 overlap models (AIM, ARF, AutoDiff, BayesianNetwork, CART, DPCART, NFlow, SMOTE, TabDDPM, TVAE) and 6 capability dimensions, we optimize $c_{g,j} \in \{0, 1, 2, 3, 4\}$ as integer parameters.

Scale factors (18 parameters). For each of the 3 intents (privacy, fidelity, utility) and 6 capability dimensions, we optimize $\alpha_{i,j} \in [0.0, 10.0]$ as continuous parameters.

This yields a total of $10 \times 6 + 3 \times 6 = 78$ parameters.

A.3 OBJECTIVE FUNCTION

We maximize the average Spearman rank correlation between predicted and oracle rankings on the training split (12 dataset–intent pairs):

$$\max_{\mathbf{c}, \boldsymbol{\alpha}} \frac{1}{|\mathcal{T}_{\text{train}}|} \sum_{(d,i) \in \mathcal{T}_{\text{train}}} \rho\big(\hat{\sigma}(d, i; \mathbf{c}, \boldsymbol{\alpha}),\ \sigma^*(d, i)\big) \tag{4}$$

where $\hat{\sigma}$ is the predicted ranking (sorted by $S(g, d, i)$), σ^* is the oracle ranking from ground truth, and ρ is Spearman's rank correlation. Only models present in both rankings contribute to ρ.

Spearman correlation was chosen over Top-1 accuracy because it provides a denser gradient signal: Top-1 is a binary 0/1 reward that does not distinguish between second place and last place, while Spearman rewards getting the full ordering correct.

A.4 OPTIMIZATION CONFIGURATION

We use the Tree-structured Parzen Estimator (TPE) sampler (Bergstra et al., 2011) implemented in Optuna:

- **Trials:** 500 (50 random startup trials, 450 TPE-guided)

- **Seed:** 42 (reproducible)
- **Direction:** maximize
- **Runtime:** $\sim$35 seconds on a single CPU core

During each trial, the engine's capability registry is temporarily overwritten with the sampled values. The engine then runs its standard scoring pipeline (hard filters $\rightarrow$ stress profiling $\rightarrow$ capability matching $\rightarrow$ scale-factor-weighted scoring) on all training pairs.

A.5 RESULTS

Table 4 summarizes performance on both splits.

Table 4: Joint optimization results on train and test splits.

Split	Top-1	Top-3	Spearman	NDCG
Training (12 pairs)	.250	.750	.610	.897
Test (9 pairs)	.000	.556	.557	.893

A.6 OPTIMIZED CAPABILITY SCORES

Table 5 shows the learned capability scores alongside the original hand-crafted values for all 10 models.

Table 5: Capability scores: hand-crafted (left) vs. optimized (right). Changes of ≥ 2 are bolded.

	Skew		Cardinal.		Zipfian		Small data		Correl.		Privacy	
Model	Orig	Opt	Orig	Opt	Orig	Opt	Orig	Opt	Orig	Opt	Orig	Opt
AIM	3	2	0	0	1	**3**	2	**4**	3	**0**	4	**2**
ARF	2	**4**	4	**2**	3	4	4	**0**	4	**1**	0	1
AutoDiff	1	**4**	3	**1**	2	3	2	**0**	1	**3**	0	**4**
BayesianNet.	3	4	4	4	2	2	4	4	3	2	0	**4**
CART	3	4	4	**2**	2	2	4	4	4	3	0	**4**
DPCART	2	3	0	0	2	**4**	2	1	3	**0**	3	**0**
NFlow	2	**4**	4	**2**	2	**0**	4	**0**	1	1	0	0
SMOTE	3	2	4	**2**	2	2	4	**1**	4	4	0	**3**
TabDDPM	1	2	2	2	2	**0**	2	**0**	3	2	0	**2**
TVAE	2	**0**	4	**2**	1	2	3	2	4	**2**	0	**3**

A.7 OPTIMIZED SCALE FACTORS

Table 6 shows the learned intent-conditioned scale factors.

Table 6: Optimized scale factors $\alpha_{i,j}$ per intent and capability dimension.

Intent	Skew	Cardinal.	Zipfian	Small	Correl.	Privacy
Privacy	5.69	3.78	7.32	9.83	8.90	9.10
Fidelity	2.43	7.36	4.10	3.11	2.78	9.44
Utility	0.68	2.59	3.71	5.36	1.42	2.96

Interpretation caveat. Some scale factor values reflect spurious correlations in the small training set rather than meaningful intent-capability relationships. For example, the fidelity intent assigns its highest weight (9.44) to the privacy/DP dimension. This occurs because models that happen to score well on fidelity in the training data (e.g., CART, BayesianNetwork) also receive high optimized privacy capability scores, creating a coincidental correlation that the optimizer exploits. Such

patterns are unlikely to generalize to benchmarks with different model pools or larger dataset collections. The scale factors should be interpreted as dataset-specific optimization artifacts rather than universal intent–capability mappings.

B Intent-Aware Scoring Design

The intent-conditioned scoring mechanism in SYNTHONY operates through multiplicative scale factors that reweight the base capability matching scores. This section details the design rationale.

B.1 Base Scoring

For each eligible model g and required capability j, the base match score is:

$$m_j(c_{g,j}, r_j) = \begin{cases} 1.0 & \text{if } c_{g,j} \geq r_j \\ 0.7 & \text{if } c_{g,j} = r_j - 1 \\ 0.4 & \text{if } c_{g,j} = r_j - 2 \\ 0.0 & \text{otherwise} \end{cases}$$

For non-required capabilities (where $r_j = 0$), when scale factors are active, the match score is $c_{g,j}/4$ to differentiate models by raw capability strength.

B.2 Scale Factor Application

Under an intent i, the weighted contribution of capability j to model g's total score is:

$$\text{contribution}_j = m_j \cdot w_j \cdot \alpha_{i,j}$$

where $w_j = 1.0$ if the dataset requires capability j (i.e., $r_j > 0$) and $w_j = 0.1$ otherwise. The total score is $S(g) = \sum_j \text{contribution}_j$.

B.3 Design Decisions

Bypassing the hard-problem path. When scale factors are provided, the engine's "hard problem" routing (which forces selection of GReaT for datasets with simultaneous skew + cardinality + Zipfian stress) is bypassed. This allows the scale-factor-weighted scoring to make the selection decision, which is essential for the optimizer to explore the full scoring space.

Bypassing tie-breaking. Similarly, the heuristic tie-breaking rules (e.g., "prefer ARF for small data") are skipped when scale factors are active, since the scale factors already encode user preference through learned weights.

Integer capability scores. We constrain capability scores to integers $\{0, \ldots, 4\}$ rather than continuous values. This preserves interpretability (scores map to qualitative levels: 0=none, 1=poor, 2=moderate, 3=good, 4=excellent) and reduces the effective search space, improving sample efficiency of the Bayesian optimizer.

C System Architecture

This appendix describes SYNTHONY's implementation as a deployable system, complementing the algorithmic description in the main text (Sections 3–4).

C.1 System Overview

SYNTHONY is organized into three packages that form a data-processing pipeline (Figure 1): (1) **Data Infrastructure** profiles raw datasets into stress vectors, (2) **Model Core** maintains the capability registry and scoring logic, and (3) **Orchestration** combines profiles with registry scores to produce ranked recommendations. Three interface layers—a REST API, an MCP server, and a CLI—expose this pipeline to different consumer types (human users, AI agents, and scripts, respectively).

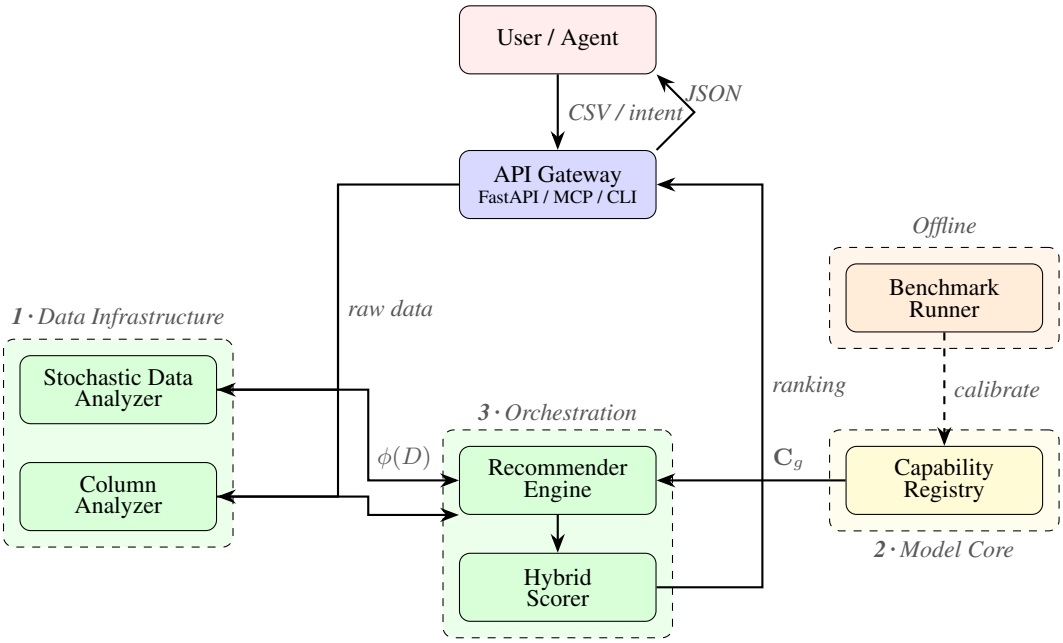

Figure 1: SYNTHONY system architecture. Solid arrows denote the real-time recommendation pipeline; dashed arrows denote the offline calibration loop (Section 3.4). Data flows left through profiling, center through scoring, and returns right through the API.

Package 1: Data Infrastructure. The `StochasticDataAnalyzer` applies four detectors—skewness, cardinality, Zipfian concentration, and data-size classification—to produce a stress profile $\phi(D) \in \mathbb{R}^4$ (Section 3.2). The `ColumnAnalyzer` computes per-column difficulty scores (type inference, missing-value density, cardinality ratio) that provide fine-grained context for the LLM-based recommendation path.

Package 2: Model Core. The capability registry (`model_capabilities.json`, v7.0.0) stores 15 synthesizer specifications, each containing: (i) capability scores $\mathbf{c}_g \in \{0, \ldots, 4\}^6$ across six dimensions, (ii) empirical quality metrics from Spark benchmarks (10 datasets, 14 models), (iii) constraint metadata (GPU requirements, min/max row limits, DP certification), and (iv) engine configuration (score decay curve, tie-breaking priorities, hard-problem routing).

Package 3: Orchestration. The `ModelRecommendationEngine` implements the full selection pipeline described in Algorithm 1: hard filtering $\rightarrow$ stress profiling $\rightarrow$ capability matching $\rightarrow$ scale-factor-weighted scoring $\rightarrow$ tie-breaking. Three recommendation methods are supported: `rule_based` (deterministic, sub-second), `llm` (OpenAI/vLLM-backed, 5–30s), and `hybrid` (rule-based candidates re-ranked by LLM).

C.2 Recommendation Scoring Pipeline

Figure 2 illustrates the rule-based scoring pipeline as a decision flowchart. The pipeline processes each dataset-intent pair through seven sequential stages, with two early-exit paths (hard filters and hard problem routing).

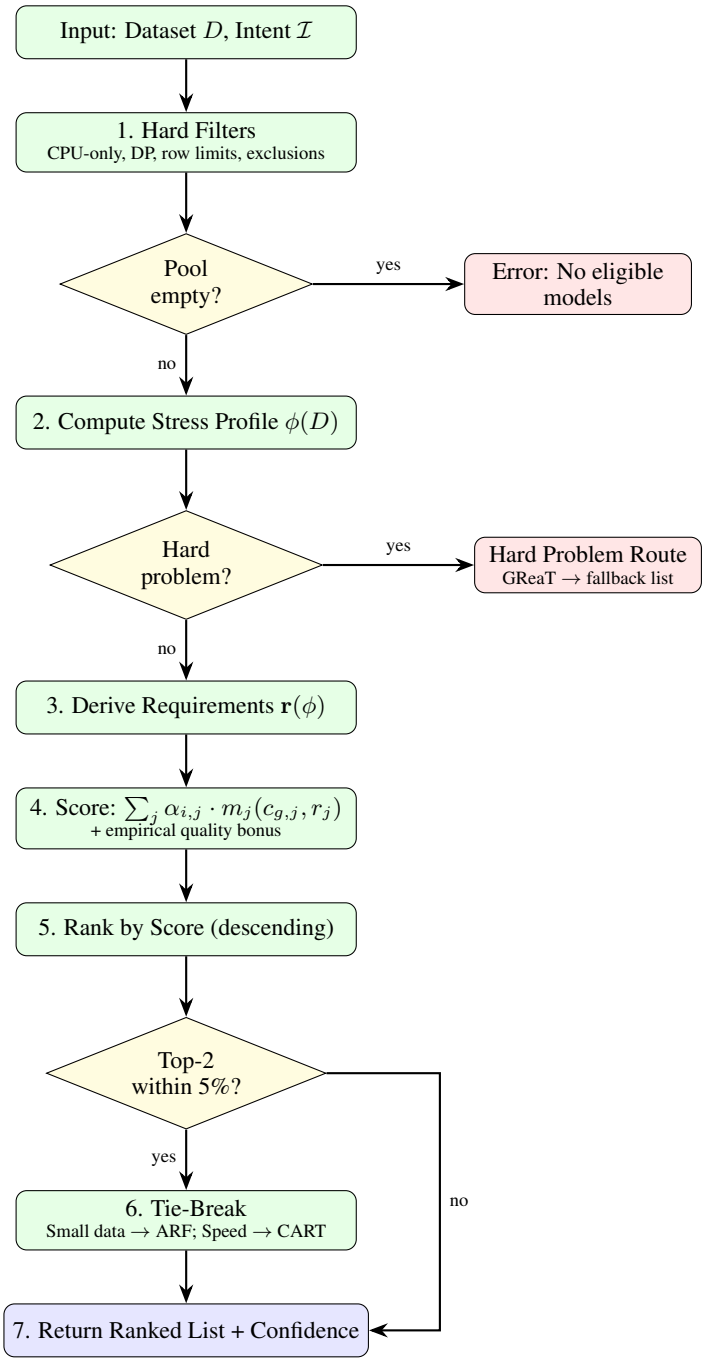

Figure 2: Rule-based scoring pipeline. The match function m_j implements a decay curve: 1.0 (exact match), 0.7 (one level below), 0.4 (two levels below), 0.0 (otherwise). Scale factors $\alpha_{i,j}$ are applied when an intent is specified. The hard-problem path is bypassed when scale factors are provided to allow the optimizer full control.

Stage 1: Hard Filters. Models are excluded based on static constraints: `exclude` flag (baselines like Identity), `cpu_only` compatibility (removes 6 GPU-required models), `strict_dp` certification (retains only PATECTGAN, AIM, DPCART with privacy_dp ≥ 3), and row-count limits (`min_rows`, `max_recommended_rows`).

Stage 2: Hard-Problem Detection. A dataset is classified as a "hard problem" when *all three* stress conditions co-occur: severe skew ($|\text{skewness}| > 2.0$), high cardinality (> 500 unique values), and Zipfian concentration (top-20% categories $> 5\%$ of mass). Hard problems are routed directly to a designated model (GReaT, or TabDDPM for large datasets), bypassing the general scoring path.

Stages 3–4: Requirement Derivation and Scoring. Each stress factor maps to a required capability level via threshold-based quantization (Section 3.2). The match function applies the score decay curve (Table 7), weighted by intent-conditioned scale factors when provided. An empirical quality bonus (avg_quality_score $\times 0.3$) from Spark benchmarks is added to each model's total score.

Table 7: Score decay curve: match score as a function of capability gap.

Condition	Match Score	Interpretation
$c_{g,j} \geq r_j$	1.0	Capability meets or exceeds requirement
$c_{g,j} = r_j - 1$	0.7	Near miss (one level below)
$c_{g,j} = r_j - 2$	0.4	Moderate gap (two levels below)
$c_{g,j} < r_j - 2$	0.0	Insufficient capability

Stages 5–7: Ranking, Tie-Breaking, and Output. Models are ranked by descending total score. When the top two models score within 5%, tie-breaking rules apply in priority order: (1) small datasets ($< 1{,}000$ rows) prefer ARF; (2) speed-optimized intents prefer CART; (3) quality-optimized intents prefer diffusion models (GPU) or tree models (CPU). The final output includes the primary recommendation with a confidence score, up to N alternatives, and a difficulty summary.

C.3 API Implementation and Persistence Layer

SYNTHONY exposes a REST API via FastAPI with a SQLite persistence layer that caches dataset profiles, tracks user sessions, and versions system prompts (Figure 3).

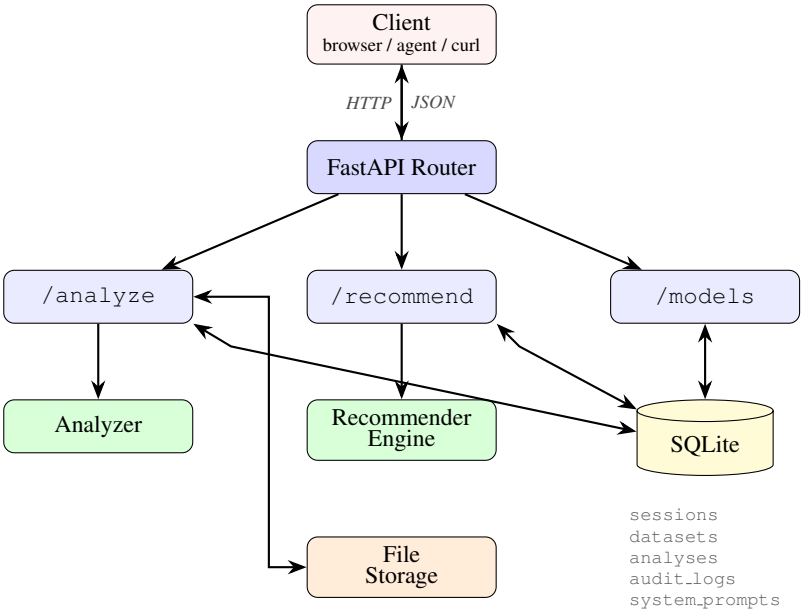

Figure 3: API architecture with SQLite persistence. Double arrows indicate read/write paths. Uploaded files are stored on disk; profiles and recommendations are cached in SQLite for session-based retrieval.

Database Schema. The persistence layer uses five SQLAlchemy ORM tables (Table 8). Sessions have a 30-day retention policy with automatic cleanup. Analyses cache serialized `DatasetPro-`

file and ColumnAnalysisResult as JSON, enabling the /recommend endpoint to retrieve cached profiles by analysis_id without reprocessing.

Table 8: SQLite database schema. PK = primary key, FK = foreign key.

Table	Key Columns	Relationships	Purpose
sessions	session_id (PK), ip_address, expires_at	→ datasets (1:N)	Track client sessions (30-day TTL)
datasets	dataset_id (PK), session_id (FK), file_path	→ analyses (1:N)	Uploaded file metadata
analyses	analysis_id (PK), dataset_id (FK), profile_json	→ system_prompts	Cached profiles + recommendations
system_prompts	prompt_id (PK), version, is_active	—	Versioned LLM system prompts
audit_logs	log_id (PK), session_id, action	—	Request audit trail

Request Flow. A typical POST /analyze-and-recommend request follows this path:

1. **Session creation:** Generate UUID; store client IP and user-agent with 30-day expiry.
2. **File persistence:** Save uploaded CSV/Parquet to data/uploads/{session_id}/, enforcing per-file (100 MB), per-session (500 MB), and total (10 GB) storage quotas.
3. **Profiling:** Run StochasticDataAnalyzer.analyze(df) and ColumnAnalyzer.analyze(df) to produce the stress profile and column analysis.
4. **Cache:** Serialize results as JSON and insert into analyses table. Subsequent /recommend calls can reference the cached analysis_id to skip reprofiling.
5. **Recommendation:** Pass the profile to ModelRecommendationEngine.recommend() with the specified method and constraints.
6. **Audit:** Log the action, endpoint, IP, and success/failure to audit_logs.

Endpoint Summary. Table 9 lists the primary API endpoints.

Table 9: REST API endpoint summary.

Method	Endpoint	Description
POST	/analyze	Upload CSV/Parquet → stress profile + column analysis (cached)
POST	/recommend	Profile → ranked recommendation (rule/LLM/hybrid)
POST	/analyze-and-recommend	One-shot: upload → profile → recommendation
GET	/models	List models with optional filters (type, CPU, DP)
GET	/models/{name}	Detailed model capabilities and constraints
GET	/health	Component status (analyzer, recommender, LLM availability)

C.4 MCP Server: Agentic Integration

SYNTHONY implements a Model Context Protocol (MCP) server that enables AI agents (e.g., Claude Code, LangChain) to invoke profiling and recommendation tools programmatically via JSON-RPC 2.0 over stdio transport (Figure 4).

MCP Primitives. The MCP specification defines three primitive types that map naturally to SYNTHONY's capabilities:

- **Tools** are executable functions that the agent can invoke with structured arguments. SYNTHONY registers 13 tools spanning data loading (list_datasets, load_dataset), profiling (analyze_stress_profile), recommendation (rank_models_hybrid, rank_models_rule, rank_models_llm), model

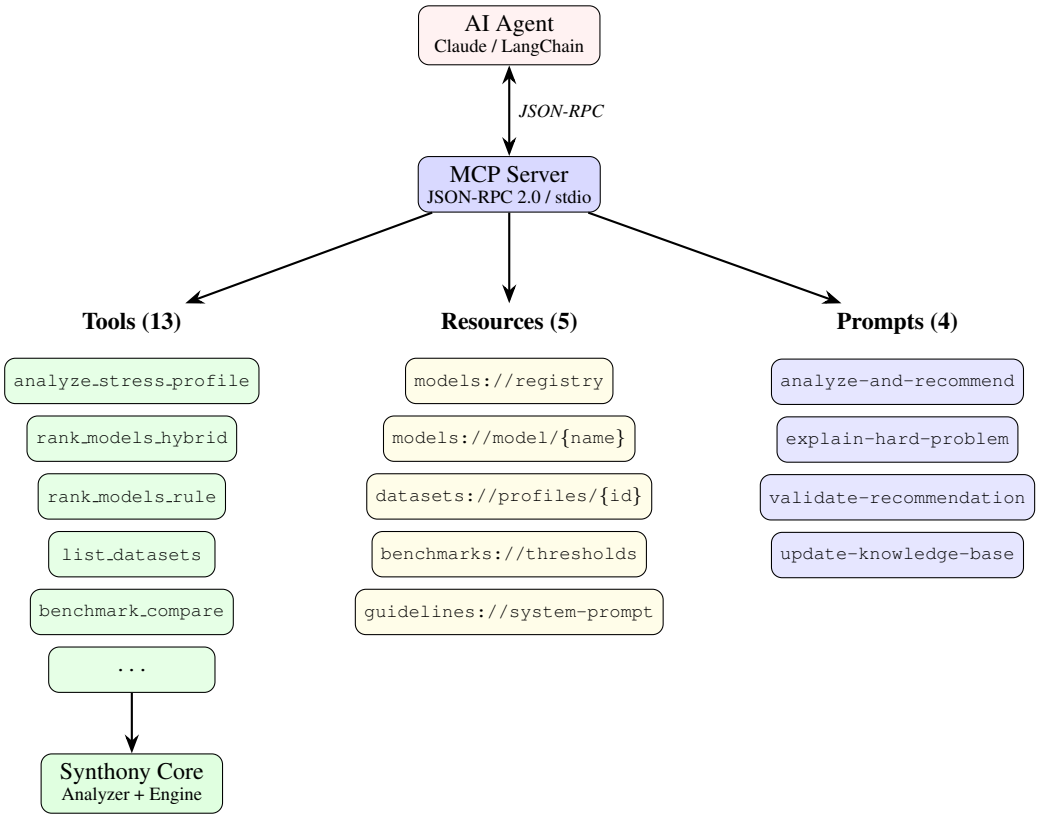

Figure 4: MCP server architecture. The server exposes SYNTHONY's capabilities through three MCP primitives: *Tools* (executable functions), *Resources* (read-only data), and *Prompts* (guided workflows). Communication uses JSON-RPC 2.0 over stdio.

inspection (`get_model_info`, `list_models`, `check_model_constraints`), explanation (`explain_recommendation_reasoning`, `get_tie_breaker_logic`), and benchmarking (`benchmark_compare`, `generate_benchmark_dataset`).

- **Resources** are read-only data endpoints identified by URIs. The server exposes the full model registry (`models://registry`), individual model details (`models://model/{name}`), cached dataset profiles (`datasets://profiles/{id}`), stress detection thresholds (`benchmarks://thresholds`), and the active LLM system prompt (`guidelines://system-prompt`).
- **Prompts** are guided multi-step workflows. Four prompts encode common task sequences: `analyze-and-recommend` (full pipeline from data path to explanation), `explain-hard-problem` (deep dive into stress factors), `validate-recommendation` (offline benchmark validation), and `update-knowledge-base` (empirical feedback loop to refine capability scores).

Agentic Workflow. An AI agent interacting with SYNTHONY through MCP follows a typical three-step workflow:

1. **Discovery:** The agent calls `tools/list` and `resources/list` to discover available capabilities.
2. **Profiling:** The agent invokes `analyze_stress_profile` with a dataset path, receiving the stress profile and column analysis as structured JSON.

3. **Selection:** The agent passes the profile to `rank_models_hybrid`, optionally specifying constraints (`cpu_only`, `strict_dp`) and the number of alternatives. The response includes the ranked recommendation, confidence scores, and decision reasoning.

This workflow can be orchestrated by a prompt template (`analyze-and-recommend`) or composed ad hoc by the agent.

Example MCP Commands. Listing 1 shows representative JSON-RPC commands for the three most common operations.

Listing 1: Example MCP JSON-RPC commands.

```
// 1. List available datasets
{"jsonrpc":"2.0", "method":"tools/call", "id":1,
 "params":{"name":"list_datasets",
   "arguments":{"format_filter":"csv"}}}

// 2. Profile a dataset
{"jsonrpc":"2.0", "method":"tools/call", "id":2,
 "params":{"name":"analyze_stress_profile",
   "arguments":{"dataset_name":"Bean"}}}

// 3. Get ranked recommendations
{"jsonrpc":"2.0", "method":"tools/call", "id":3,
 "params":{"name":"rank_models_hybrid",
   "arguments":{
     "dataset_profile": <output from step 2>,
     "method":"hybrid", "top_n":3}}}

// 4. Read model registry (resource)
{"jsonrpc":"2.0", "method":"resources/read", "id":4,
 "params":{"uri":"models://registry"}}

// 5. Invoke guided workflow (prompt)
{"jsonrpc":"2.0", "method":"prompts/get", "id":5,
 "params":{"name":"analyze-and-recommend",
   "arguments":{"data_path":"/data/insurance.csv"}}}
```

Self-Correcting Feedback Loop. The `update-knowledge-base` prompt enables a feedback cycle: after validating a recommendation against benchmark results, the agent can propose updates to the capability registry or system prompt. This loop is currently human-supervised (the agent proposes changes; a developer reviews and applies them), but the architecture supports fully autonomous calibration via the offline Bayesian optimization described in Appendix A.

D Implementation Details

We standardize the training and evaluation interface across all candidate synthesizers via a unified API exposing `fit()`, `sample()`, and `get_hyperparameter_space()` methods, enabling controlled comparisons under identical preprocessing and sampling conditions. An evaluation harness computes a metric vector $\mathbf{m} \in \mathbb{R}^d$ spanning statistical fidelity (e.g., column shape score), privacy risk (e.g., proportion of synthetic records closer to real data than holdout records), and utility (downstream ML task performance, measured by Test ROC AUC for classification and Test Adjusted R^2 for regression). An LLM can optionally translate natural-language intent into a scalarized objective, but the core selection is performed by the stress-aware scoring function described in Section 3.

E Benchmark Datasets

We provide the URL for the sources of each downstream benchmark set considered in the paper.

1. **Abalone** (OpenML) : `https://www.openml.org/search?type=data&sort=runs&id=183&status=active` (multi-class)
2. **Bean** (UCI) : `https://archive.ics.uci.edu/dataset/602/dry+bean+dataset` (Multi class)

3. **faults** (UCI) : https://archive.ics.uci.edu/dataset/198/steel+plates+faults (multi-class)
4. **IndianLiverPatient** (Kaggle) : https://www.kaggle.com/datasets/uciml/indian-liver-patient-records (binary)
5. **insurance** (Kaggle) : https://www.kaggle.com/datasets/mirichoi0218/insurance (Regression)
6. **Obesity** (Kaggle) : https://www.kaggle.com/datasets/tathagatbanerjee/obesity-dataset-uci-ml (multi-class)
7. **wilt** (OpenML) : https://www.openml.org/search?type=data&sort=runs&id=40983&status=active (binary)

F Evaluation Metrics

The three intent-specific metrics used to construct oracle rankings in Section 4 are as follows.

F.1 Fidelity: Column Shape Score

We measure fidelity via the **Column Shape Score**, which evaluates similarity between real and synthetic marginal distributions on a per-column basis. For numerical columns, we use the Kolmogorov–Smirnov (KS) test statistic subtracted from one; for categorical columns, we use one minus the Total Variation Distance (TVD). The final score is the average across all columns, with higher values indicating better fidelity.

F.2 Utility: Downstream ML Performance

We evaluate machine learning utility by training classifiers or regressors on synthetic data and evaluating on held-out real test data. For classification datasets we report **test ROC AUC**; for regression datasets we report **test adjusted $\mathbf{R}^2$**. Higher values indicate better utility in both cases.

F.3 Privacy: Proportion Closer to Real

We evaluate privacy risk via **Proportion Closer to Real** (Platzer & Reutterer, 2021), which measures the fraction of synthetic records whose nearest real neighbor is in the training set rather than the holdout set. Lower values indicate better privacy (less memorization risk). For oracle ranking construction, we invert this metric so that higher rank corresponds to better privacy.

G Code and Data Availability

The complete source code for Synthony, including the profiling pipeline, recommendation engine, capability registry, optimization scripts, REST API, and MCP server, is publicly available at:

https://github.com/UCLA-Trustworthy-AI-Lab/Synthony

And, the source code for the Synthesizer has refers to as:

https://github.com/UCLA-Trustworthy-AI-Lab/table-synthesizers

The repository is organized as follows:

Table 10: Repository structure and correspondence to paper sections.

Directory / File	Contents	Paper Reference
`src/synthony/core/`	Data loaders, `StochasticDataAnalyzer`, schemas	Section 3.2
`src/synthony/detectors/`	Skewness, cardinality, Zipfian, correlation detectors	Section 3.2
`src/synthony/recommender/`	`ModelRecommendationEngine`, scoring logic	Section 3.3
`config/model_capabilities.json`	Capability registry (v7.0.0, 15 models $\times$ 6 dims)	Section 3.3
`scripts/optimize_scaling.py`	Bayesian optimization of scale factors (Optuna/TPE)	Section A
`src/synthony/api/`	FastAPI REST server with SQLite persistence	Appendix C.3
`mcp_server/`	MCP server (JSON-RPC 2.0 / stdio)	Appendix C.4
`src/synthony/benchmark/`	Data quality metrics (KL/JS divergence, fidelity, privacy)	Section 4
`tests/`	Unit and integration test suites	—

Reproducing Results. To reproduce the experiments reported in this paper:

```
# 1. Install Synthony with all dependencies
pip install -e ".[all]"

# 2. Profile a dataset
synthony-profile dataset/input_data/Bean.csv --verbose

# 3. Run rule-based recommendation
synthony-recommender -i dataset/input_data/Bean.csv --method rulebased

# 4. Run the Bayesian optimization (scale factors)
python scripts/optimize_scaling.py

# 5. Start the API server
uvicorn synthony.api.server:app --reload

# 6. Start the MCP server (for AI agent integration)
python -m mcp_server.server --verbose
```

Full installation instructions, environment setup, and dataset preparation steps are documented in the repository's `README.md`. The benchmark datasets (Abalone, Bean, IndianLiverPatient, Obesity, faults, insurance, wilt) are sourced from OpenML (Vanschoren et al., 2013).