# Towards Automated Causal Effect Estimation with Self-Evolving AI

**Can Wang** [1] **Hongyu Zhao** [2] **Yiqun Chen** [1 2]

## Abstract

Causal inference is central to scientific discovery, yet choosing appropriate methods remains challenging due to the complexity of statistical methodology and real-world data. Inspired by the success of artificial intelligence in accelerating scientific software, we introduce an evolutionary framework that uses large language models (LLMs) to discover and iteratively refine causal estimators. Across four semi-synthetic causal benchmarks, LLM-guided evolutionary search discovers estimator programs whose upper tail improves over strong fixed baselines, including Pareto-frontier performance relative to 58 ACIC 2022 human submissions. We further study proxy-guided search when true effects are unavailable, finding strong proxy fidelity on some benchmarks and failure on others. Analysis of evolutionary trajectories shows that the system rediscovered and recombined known causal-estimation components in benchmark-specific ways. Our findings suggest that LLM-guided evolution could support methodological search when rewards are only partly observed, as in causal inference.

## 1. Introduction

Causal inference answers whether and how an intervention affects an outcome and has been central to scientific discovery and evidence-based decision-making (Hernán & Robins, 2020; Pearl, 2009; Bailey et al., 2024). One of the fundamental challenges in causal inference is the *counterfactual problem*: for any unit receiving treatment, we cannot observe what would have happened under control, and vice versa (Imbens & Rubin, 2015). Practitioners must therefore rely on assumptions and statistical methods to estimate treatment effects from observational data, where treatment assignment may depend on observed and unobserved confounders (Rosenbaum & Rubin, 1983). Despite decades of methodological progress—from propensity score methods (Stuart, 2010; Austin, 2011) to doubly robust estimators (Chernozhukov et al., 2018; Kennedy, 2024; Bang & Robins, 2005) and causal machine learning (Feuerriegel et al., 2024)—two persistent challenges remain. First, *which estimator to use*: selecting the right approach requires substantial expertise about both the statistical estimators and the substantive features of the data. Second, *how to evaluate estimators*. Since ground-truth treatment effects are unobservable in real data, researchers routinely rely on *semi-synthetic benchmarks with known data-generating processes* to assess estimators (Hill, 2011; Dorie et al., 2019; Thal & Finucane, 2023).

For a concrete example, consider estimating the effect of a job training program on subsequent earnings. Each individual $i$ has potential outcomes $Y_i(1)$ and $Y_i(0)$ under treatment and control, but only one is observed. Stakeholders are often interested in the average treatment effect $\text{ATE} = \mathbb{E}[Y(1) - Y(0)]$, which requires estimating the missing potential outcomes. A human expert would typically proceed by: identifying the causal estimand (i.e., ATE in this case) under specific assumptions; choosing an estimation strategy (e.g., inverse propensity weighting, outcome regression, or doubly robust methods); selecting the models used to estimate nuisance parameters; and constructing confidence intervals. Each step involves modeling choices that can substantially affect the final estimate. To evaluate the quality of this pipeline, researchers often use semi-synthetic datasets that mimic the real-world settings they care about, and compute quantitative metrics such as bias, efficiency, and calibration of uncertainty (Imbens & Rubin, 2015).

The causal inference workflow is well-suited to automation for two reasons. First, the space of causal estimation strategies is combinatorially large but structured—different choices of identification strategy, nuisance models, and aggregation schemes can be composed programmatically. Second, the standard evaluation paradigm in causal inference already relies on simulated datasets with known ground truth, providing a natural fitness signal for optimization. Recent work has demonstrated the potential of large language model (LLM)-based agents for scientific tasks (Gao et al.,

[1]Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health, Baltimore, Maryland, USA [2]Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, USA. Correspondence to: Yiqun Chen <yiqunc@jhu.edu>.

*Proceedings of the $43^{rd}$ International Conference on Machine Learning*, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

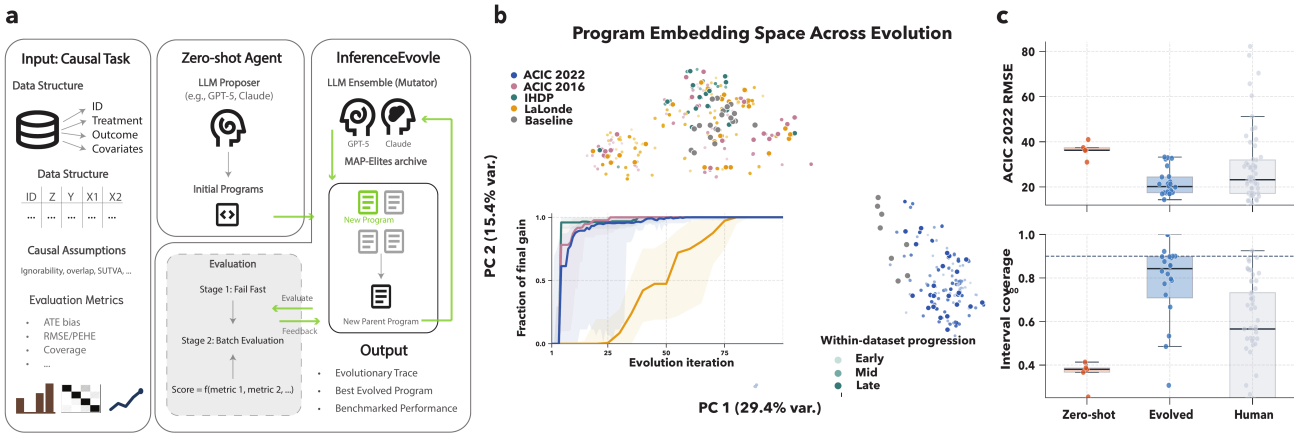


*Figure 1.* **Overview of** `InferenceEvolve` **for causal estimator discovery. a**, We start with a zero-shot program, which is iteratively improved by an LLM ensemble using benchmark-based feedback. **b**, Principal-component view of programs across evolutions (Colors = benchmarks and darker points = later checkpoints). The line graph summarizes best-so-far progress. **c**, Evolved programs compare favorably with human submissions on ACIC 2022 (Thal & Finucane, 2023). Top, RMSE distributions (lower better); bottom, empirical 90% interval coverage.

2024; Gottweis et al., 2025; Yamada et al., 2025; Swanson et al., 2025). In parallel, evolutionary code generation frameworks such as Autoresearch (Karpathy, 2026), AlphaEvolve (Novikov et al., 2025) and OpenEvolve (Sharma, 2025) have shown that LLMs can discover state-of-the-art algorithms through iterative program improvement guided by automated evaluation.

**Related work:** Recent LLM-based and automated work in causal inference has primarily evaluated or automated pieces of the causal-analysis workflow rather than searching over new treatment-effect estimators. CauSciBench evaluates whether LLMs can identify variables, select and implement an appropriate causal method, and report an effect estimate; Causal AI Scientist, CATE-B, OpportunityFinder, CausalTune, and AutoCATE automate causal analysis by selecting adjustment sets, methods, estimators, or CATE pipelines from predefined workflow/search spaces; and NATURAL uses LLMs to extract or impute variables from unstructured text so that classical estimators can be applied (Acharya et al., 2025; Verma et al., 2025; Berrevoets et al., 2025; Nguyen et al., 2023; Kraev et al., 2022; Vanderschueren et al., 2025; Dhawan et al., 2024). AutoCATE is closest in spirit because it extensively validates doubly robust proxy scores for selecting among predefined CATE base learners; our setting instead searches executable estimator programs and studies comparisons with human submissions, convergence, and failure cases. Adjacent work also uses LLM-guided evolution for causal structure learning, but the object being discovered is a graph score function rather than a treatment-effect estimator (Wang et al., 2025). Thus, existing systems either benchmark causal reasoning, automate method/variable/pipeline selection, or adapt fixed estimators to non-tabular data. To our knowledge, open-ended treatment-effect estimator discovery—where the agent improves the estimator itself by jointly searching over nuisance models, weighting or meta-learning structure, and interval construction—has not been explored.

**Contribution:** Motivated by this gap, we introduce `InferenceEvolve`, an evolutionary framework that represents treatment-effect estimators as executable programs, evaluates them on benchmark tasks, and iteratively refines them through LLM-guided mutations (Figure 1). First, we define an open-ended estimator-discovery setting across ACIC 2022 (Thal & Finucane, 2023), ACIC 2016 (Dorie et al., 2019), IHDP (Hill, 2011), and LaLonde benchmark (LaLonde, 1986; Dehejia & Wahba, 1999), spanning panel, cross-sectional, semi-synthetic, and observational designs. Because these benchmarks provide many competition or semi-synthetic replicates, each run optimizes on one subset and reports only on disjoint held-out replicates; this does not rule out benchmark memorization, but requires improvements beyond the zero-shot seed to generalize across unseen replicate draws. Second, we develop proxy objectives based on doubly robust treatment-effect estimates, enabling iterative refinement without ground-truth counterfactuals. Third, we analyze the resulting estimators across model, split, and $\lambda$ settings to guide practical use of the framework.

## 2. Methods

### 2.1. Benchmarks and evaluation metrics

We evaluated `InferenceEvolve` on four benchmark families that expose many semi-synthetic or competition replicates rather than a single fixed train/test table: ACIC 2022 Track 2 panel data (Thal & Finucane, 2023), ACIC 2016 (Dorie et al., 2019), IHDP (Hill, 2011), and

semi-synthetic LaLonde/Dehejia–Wahba samples (LaLonde, 1986; Dehejia & Wahba, 1999). The LaLonde benchmark uses observed covariates and treatment assignment from the NSW/PSID job-training setting but evaluates against generated potential-outcome columns included in the released sample files and hidden from candidate programs. For the cross-sectional benchmarks, each candidate returns individual treatment-effect estimates $\hat{\tau}_i$ and an ATE estimate, and we report held-out precision in estimation of heterogeneous effects

$$\sqrt{\mathrm{PEHE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{\tau}_i - \tau_i)^2}, \qquad (1)$$

together with absolute ATE error. For ACIC 2022, whose target is a replicate-level patient-weighted sample average treatment effect on the treated, we report held-out RMSE across replicates and empirical 90% interval coverage. All main-text claims use the prespecified 20:80 split between search replicates and untouched held-out replicates; matched 50:50 reruns are treated as sensitivity analyses in the supplement.

### 2.2. `InferenceEvolve` search

`InferenceEvolve` treats an estimator as an executable Python program with a benchmark-specific `estimate(...)` interface. It builds on OpenEvolve (Sharma, 2025), using a MAP-Elites archive (Mouret & Clune, 2015) to retain high-scoring and diverse programs while an LLM ensemble proposes full-program mutations. The archive uses OpenEvolve's built-in complexity and code-diversity descriptors; implementation details are in Supplementary Section S1.5. Search-time scoring is computed only on training replicates, and the archived final program is evaluated once on held-out replicates; the full workflow is summarized in Algorithm 1.

For search-time selection, ACIC 2022 uses $S_{\text{ACIC2022}} = [1 + \log(1 + \mathrm{RMSE}_{\mathcal{D}}) + \lambda \log(1 + |\mathrm{Coverage}_{\mathcal{D}} - 0.9|/0.9)]^{-1}$, where $\mathrm{RMSE}_{\mathcal{D}}$ and $\mathrm{Coverage}_{\mathcal{D}}$ are aggregate metrics over the evaluated replicates. IHDP, ACIC 2016, and LaLonde use $S_{\text{ITE}} = [1 + \log(1 + \overline{\sqrt{\mathrm{PEHE}}}) + \lambda \log(1 + \overline{|\mathrm{ATE\ error}|})]^{-1}$, with overlines denoting averages over search replicates. Scores computed on the training data are used to guide evolution; all reported comparisons use *unseen held-out data*.

### 2.3. Proxy objectives, baselines, and program analysis

To test search without hidden counterfactuals, we also ran proxy-guided evolution. For IHDP, ACIC 2016, and LaLonde, the proxy objective combines cross-fitted doubly robust pseudo-outcome fit, ATE discrepancy from a cross-fitted AIPW estimate, and normalized R-loss (Kennedy, 2023; 2024; Mahajan et al., 2024). For ACIC 2022, the proxy uses a doubly robust difference-in-differences pseudo-target with point-error, coverage, and interval-width terms. Full proxy formulas and implementation controls are given in Supplementary Section S1.

**Algorithm 1** `InferenceEvolve` workflow for causal estimator discovery

**Require:** Baseline program $P_0$, evaluator $E$, configuration $C$, benchmark replicates $\mathcal{D}$
**Ensure:** Best program $P^\star$, evolution traces
Seed all RNGs with global seed; split $\mathcal{D}$ into training $\mathcal{D}_{\text{train}}$ and held-out $\mathcal{D}_{\text{val}}$.
Initialize the MAP-Elites archive $\mathcal{A}$ with $(P_0, E(P_0, \mathcal{D}_{\text{train}}))$.
Build LLM ensemble: $\{(w_1, \text{primary model}), (w_2, \text{secondary model})\}$.
**for** $i = 1 \ldots C.$max_iterations **do**
  $(P_{\text{parent}}, \mathcal{I}) \leftarrow$ sample parent and inspiration set from $\mathcal{A}$.
  $q \leftarrow$ ComposePrompt$(P_{\text{parent}}, \mathcal{I}, \text{metrics}, \text{top-}k \text{ programs})$.
  Sample LLM $\ell$ from ensemble with probabilities $(w_1, w_2)$.
  $P' \leftarrow \ell(q)$ {Generate candidate program}
  **Stage 1 (fail-fast):** $m_1 \leftarrow E(P', \mathcal{D}^{(1)}_{\text{train}})$ on $\sim$10% of training replicates; $s_1 \leftarrow m_1[\texttt{combined_score}]$.
  **if** $s_1 < \theta_1$ **then**
    Discard $P'$; **continue**.
  **end if**
  **Stage 2 (full):** evaluate $E(P', \mathcal{D}_{\text{train}})$ on all training replicates.
  Compute composite score $S(P')$ using the dataset-specific scoring function.
  Insert $(P', S, \text{metrics})$ into $\mathcal{A}$; update MAP-Elites bins and $P^\star$.
**end for**
**Final evaluation:** run $E(P^\star, \mathcal{D}_{\text{val}})$ on held-out replicates.
**return** $P^\star$, evolution traces, evaluation logs.

We compared against a fixed suite of representative causal estimators evaluated on the same held-out replicates: ordinary least squares adjustment, inverse propensity weighting (Rosenbaum & Rubin, 1983; Austin, 2011), causal forests (Battocchi et al., 2019; Wager & Athey, 2018; Athey et al., 2019), BART (Hill, 2011), benchmark-specific ACIC 2022 panel baselines, and the 58 ACIC 2022 human competition submissions. Where a baseline wrapper used internal cross-fitting or cross-validation, we used the selected fit, and Table 1 reports the strongest held-out comparator from the baseline pool for each metric. This strongest-comparator protocol is an oracle upper bound for methods that only select among the fixed baseline pool, because some automated selectors did not converge on all datasets. The factorial sweep crossed four datasets, two primary LLMs (GPT-5 and Gemini 3 Flash Preview; Achiam et al., 2023; Team et al., 2024), three $\lambda$ values, two held-out split regimes, and two replicate seeds, yielding 24 true-score final programs per dataset; matched proxy-guided runs used the same setup.

Finally, we analyzed final programs by source-code length, LLM-translated methods length, TF-IDF and embedding similarity to seven commit-pinned official reference wrappers, and a three-model LLM-as-judge taxonomy reviewed by two graduate students with causal-inference expertise. Details of prompts, allowed libraries, hyperparameters, reference wrappers, and judge aggregation appear in Supplementary Sections S1–S2.2.

## 3. Results

We first compared the evolved estimators against quantitative baselines, including the zero-shot LLM baseline, off-the-shelf causal inference methods across all four benchmarks, and, for ACIC 2022, the distribution of 58 human competition submissions. We then qualitatively examined the learned programs, focusing on the estimator families that emerged at convergence, their overlap with existing work, and their coding and conceptual complexity.

### 3.1. `InferenceEvolve` finds strong candidate estimators

For the full true-score sweep, each dataset has 24 evolved final programs from a $2 \times 3 \times 2 \times 2$ factorial design over two primary LLMs from the GPT and Gemini families (GPT-5 and Gemini 3 Flash Preview; Achiam et al., 2023; Team et al., 2024), regularization weight $\lambda$ (larger $\lambda$ puts more weight on ATE estimation error and coverage error), held-out split (20% versus 50% training), and replicate seed. All main-text figures and tables focus on the prespecified 20:80 split; the matched 50:50 reruns are used only as sensitivity analyses in the Supplementary Information. Search received feedback only from the training replicates, whereas every quantitative claim in the main text is evaluated on the untouched held-out replicates. Proxy-guided runs used the same benchmarks but saw only proxy metrics during search, not hidden counterfactuals.

On the ACIC 2022 data with rich human competition results ($n = 58$), the 12 final estimators from different configurations of `InferenceEvolve` obtained a median root mean squared error (RMSE) 18.8 and median empirical coverage 0.84 (target level, 0.9), compared with 23.2 and 0.57, respectively, across the 58 human submissions (Fig. 1). By RMSE, 8 of the 12 evolved final estimators outperformed the median human submission and the best evolved run (RMSE 14.4) outperformed 51 of 58 submissions. The top two evolved programs also lie on the *Pareto frontier* with respect to RMSE and absolute deviation from nominal coverage (Miettinen, 1999); that is, *no human submission achieved both lower RMSE and coverage closer to the target 90% level than either of these two programs*. Without the self-evolving improvement, LLM performance was clearly weaker: for example, GPT-5 zero-shot programs produced a difference-in-differences estimator with RMSE 33.1 and coverage 0.40 (Supplementary Tables 2 and 3).

For the other semi-synthetic datasets without direct human benchmarks, we compared final programs against commonly used reference estimators spanning regression adjustment, propensity weighting, causal forests, tree-based meta-learners, and benchmark-specific panel baselines (see Methods). We display the held-out test-set metrics across all configurations and runs in Fig. 2, with detailed metrics of the best runs reported in Table 1. The main pattern is distributional: `InferenceEvolve` produced candidate estimators whose best and upper-tail held-out performance improved over fixed package baselines, while average gains varied by benchmark, guidance mode, and metric. On IHDP, the strongest baseline (BART) achieved $\sqrt{\text{PEHE}}$ 2.41 and ATE error 0.11, compared with 1.22 and 0.06 for the best true-score run; the best proxy-guided run reached a lower $\sqrt{\text{PEHE}}$ of 1.034 while yielding ATE error 0.074. On ACIC 2016, the strongest baseline (causal forest) achieved 1.28 and 0.15, compared with 0.86 and 0.09 for the best `InferenceEvolve` run. On LaLonde, the strongest package baseline (BART) achieved 0.77 for $\sqrt{\text{PEHE}}$ and 0.07 for ATE, while the design-based difference-in-means reference achieved normalized ATE error 0.05; the best true-score run reached 0.598 and 0.033, whereas the best proxy-guided run reached 0.693 and 0.054, improving over the package $\sqrt{\text{PEHE}}$ baseline but not over the stronger design-based ATE comparator. Across all three ITE benchmarks, these paired $\sqrt{\text{PEHE}}$/ATE comparisons show that the best true-score gains extended beyond average calibration to unit-level heterogeneity recovery.

The final programs produced by `InferenceEvolve` tended to be longer in code length and more complex in terms of methodology than the off-the-shelf baselines. This increase in complexity should be interpreted together with the evaluation protocol: the search never saw the held-out metrics reported in Fig. 2 and Table 1; those numbers were computed only after the final program was frozen. When GPT-5 was used to transcribe the final code into manuscript-style methods paragraphs, the resulting descriptions were consistently shortest for the baseline programs and longer for the true-evolved and proxy-evolved programs across all four datasets; the same ordering was also evident in code line counts (Fig. 2). This pattern was robust to the choice of transcription LLM, including Gemini 2.5 Pro and Claude Sonnet 4.5. Full distributions of code length, measured in characters and lines, are shown in Supplementary Figs. 2 and 1.

### 3.2. Evolution discovers dataset-specific and novel estimator families

Beyond held-out performance, it is also important to ask whether evolution reveals a structured search process rather than generic code drift. We summarize our findings in Figure 3: 1. different benchmarks converge to different estimator families; and 2. the evolved programs can be anchored to a set of benchmark-specific published methods and refinements.

We first note that the converged pool of estimators were strongly dataset-specific, which explains the distinct clusters in PCA plot in Figure 1: since it is a panel data with the same

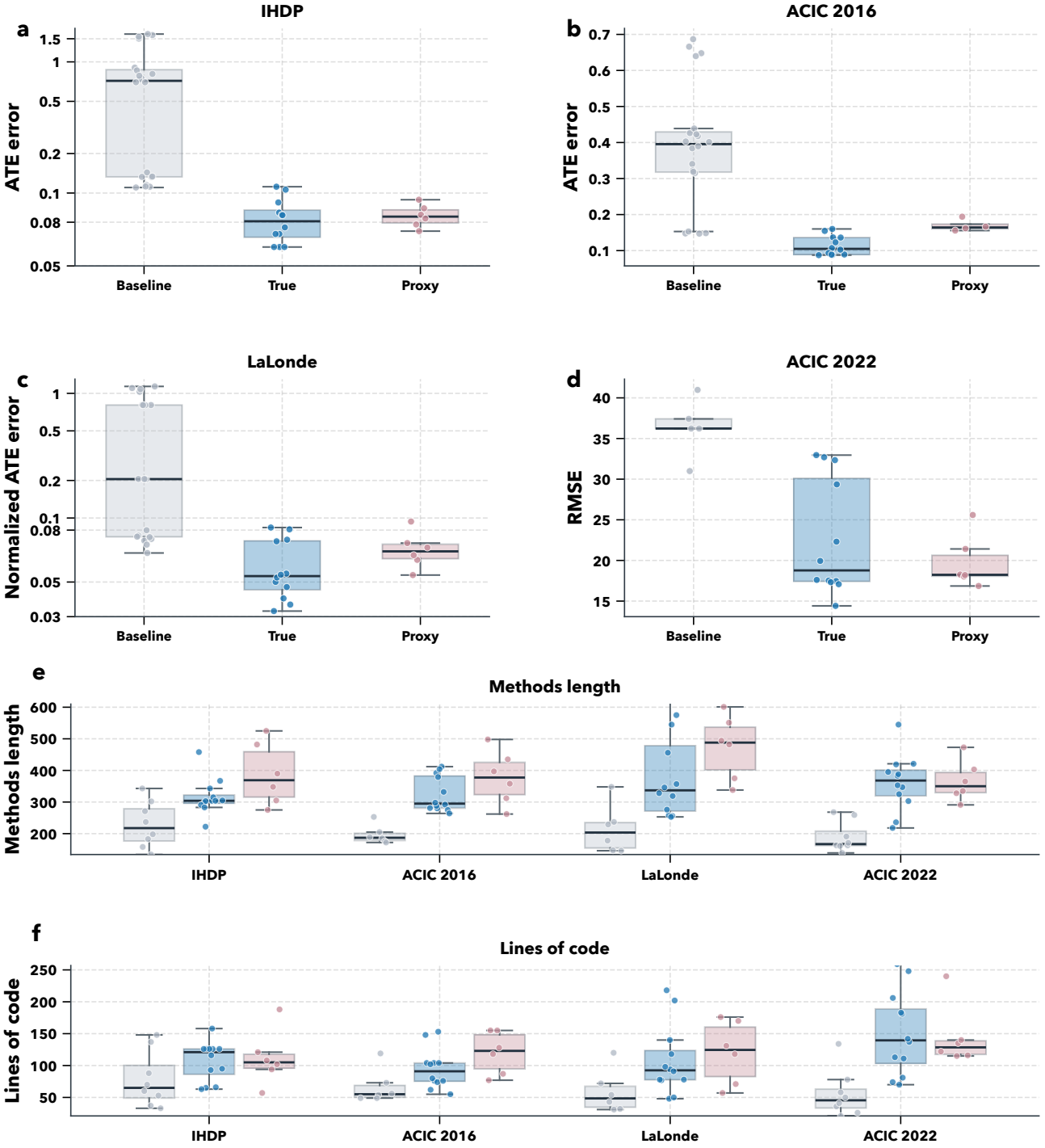


*Figure 2.* **Held-out performance distributions for baselines, true-evolved programs, and proxy-evolved programs, with text- and code-complexity summaries.** Panels a–d show the metric distributions on the held-out set for all baseline methods and the final true-evolved and proxy-evolved runs. Panels e and f summarize GPT-5 methods length and code lines by dataset.

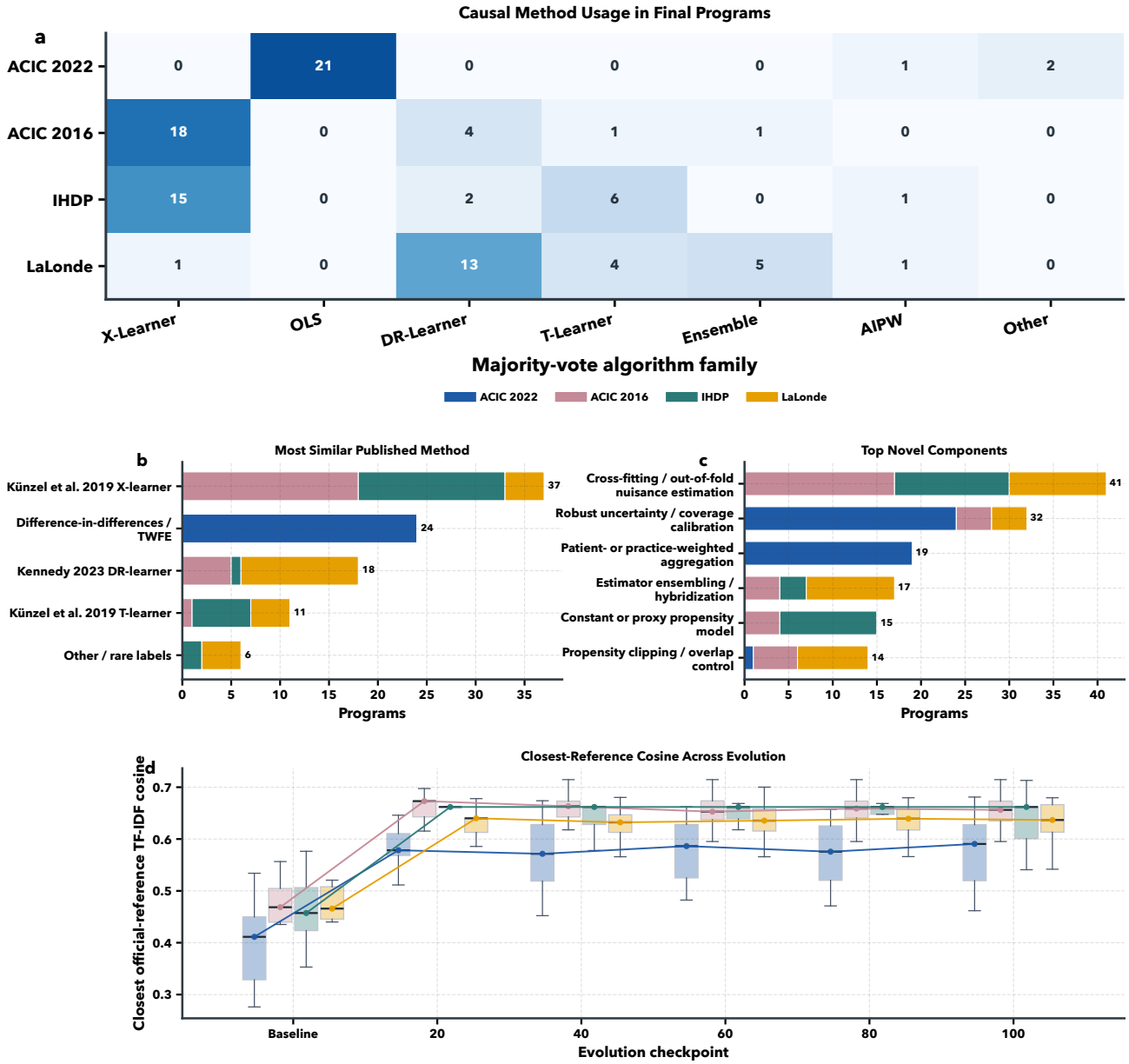


*Figure 3.* `InferenceEvolve` **converges to dataset-specific estimator families without collapsing onto a single public template. a**, Majority-vote-based algorithm-family assignments for the 96 final evolved programs; each cell reports the number of final programs from that dataset assigned to that family. **b**, Majority-vote-based most similar published method. **c**, Novel algorithmic components extracted from the union of LLM-as-a-judge reports for each program. **d**, Tokenized TF-IDF cosine similarity between each program and its closest official reference wrapper across zero-shot baselines and checkpoints, with boxplots stratified by dataset.

*Table 1.* **Best evolved estimators versus baseline methods.** The baseline column gives the candidate used for each row: the strongest package baseline in Fig. 2. *Best evolved* and *best* $\Delta$ *(%)* refer to the single best final program under each guidance mode, where positive *best* $\Delta$ indicates percent error reduction relative to the row-specific baseline comparator. *Mean* $\pm$ *SE* and *% beat* summarize the performance across all parameter settings for `InferenceEvolve`.

| Dataset | Baseline | Mode | Best | Mean $\pm$ SE | % beat | Best $\Delta$ (%) |
|---|---|---|---|---|---|---|
| IHDP | | | | | | |
| $\sqrt{\text{PEHE}}$ | 2.411 (BART) | True | 1.222 | $1.392 \pm 0.021$ | 100% | 49.3% |
| | | Proxy | **1.034** | $1.632 \pm 0.311$ | 83% | 57.1% |
| ATE error | 0.110 (BART) | True | **0.063** | $0.081 \pm 0.005$ | 100% | 42.9% |
| | | Proxy | 0.074 | $0.084 \pm 0.003$ | 100% | 33.0% |
| ACIC 2016 | | | | | | |
| $\sqrt{\text{PEHE}}$ | 1.283 (Causal forest) | True | **0.858** | $1.011 \pm 0.042$ | 100% | 33.1% |
| | | Proxy | 1.504 | $2.037 \pm 0.196$ | 0% | $-17.2\%$ |
| ATE error | 0.147 (Causal forest) | True | **0.087** | $0.114 \pm 0.011$ | 100% | 40.6% |
| | | Proxy | 0.156 | $0.170 \pm 0.008$ | 0% | $-5.7\%$ |
| LaLonde | | | | | | |
| $\sqrt{\text{PEHE}}$ | 0.765 (BART) | True | **0.598** | $0.704 \pm 0.016$ | 83% | 21.8% |
| | | Proxy | 0.693 | $0.754 \pm 0.017$ | 67% | 9.4% |
| ATE | 0.048 (Diff-in-means) | True | **0.033** | $0.057 \pm 0.004$ | 17% | 30.8% |
| | | Proxy | 0.054 | $0.070 \pm 0.005$ | 0% | $-12.6\%$ |
| ACIC 2022 | | | | | | |
| RMSE | 23.40 (Backdoor adj.) | True | **14.41** | $22.58 \pm 2.39$ | 50% | 38.4% |
| | | Proxy | 16.85 | $19.72 \pm 1.33$ | 83% | 28.0% |

units across timepoints, ACIC 2022 concentrated almost entirely on OLS / panel estimators (21/24). By contrast, ACIC 2016 converged on X-learners (Künzel et al., 2019) (18/24), IHDP on X- and T-learners (Künzel et al., 2019) (15/24 and 6/24), and LaLonde on DR-learner (Kennedy, 2023) and ensemble hybrids (13/24 and 5/24). Across all 96 final programs, the majority-vote family counts were X-learner (34), OLS (21), DR-learner (19), and T-learner (11). The three judges in the LLM-as-a-judge pipeline agreed on the coarse family label for 84/96 programs, but agreed on a single nearest named published method for only 42/96, indicating convergence at the level of method families and components rather than exact textbook replication.

Closest official-reference TF-IDF cosine increased from an overall median of 0.446 among the zero-shot baseline programs to 0.640 by checkpoint 20, then remained nearly flat through checkpoint 100 (median 0.638). This pattern is consistent with rapid movement toward familiar estimator families followed by benchmark-specific refinement. Relative to the closest official references, the most common novel components were cross-fitting or out-of-fold nuisance estimation (41/96 programs), robust uncertainty or coverage calibration (32/96), patient- or practice-weighted aggregation (19/96), estimator ensembling or hybridization (17/96), constant or proxy propensity models (15/96), and propensity clipping or overlap control (14/96). Pairwise code similarity among evolved programs again showed dataset-specific convergence: overall TF-IDF similarity was higher within datasets than between datasets (0.359 versus 0.228), and the same ordering held for `text-embedding-3-large` embeddings (0.776 versus 0.665).

These structural shifts were accompanied by larger programs and longer scientific write-ups. Across datasets, baseline programs had a median size of 54 lines and 2,004 characters, compared with 104 lines and 3,922 characters for true-evolved programs and 119.5 lines and 4,698.5 characters for proxy-evolved programs (Supplementary Fig. 1). Translating each program into a manuscript-style methods paragraph with GPT-5, Gemini 2.5 Pro, or Claude Sonnet 4.5 yielded the same ordering: median description length increased from 172.5 words for baselines to 263.0 for true-evolved programs and 279.5 for proxy-evolved programs (Fig. 2). Table 2 summarizes representative solution families relative to the closest official reference wrapper.

Search dynamics were benchmark-dependent: the median run reached 90% of its final best-so-far score by iteration 5 on IHDP, iteration 12 on ACIC 2016, and iteration 16 on ACIC 2022, whereas LaLonde improved more gradually and reached the same threshold around iteration 75 (Supplementary Fig. 6). On ACIC 2022, the strongest run moved from an under-calibrated weighted TWFE seed to an augmented DiD estimator by adding pre-period anchoring, post-period differencing, trend adjustment, and cluster-robust inference, reducing RMSE from 33.1 at initialization to 15.2 at its final best checkpoint; component-metric sensitivity analyses are reported in Supplementary Section S3.2.

### 3.3. Proxy-guided optimization

A central difficulty in causal estimator discovery is that the true reward is usually unobserved in real data. We therefore replaced hidden-truth objectives with observed-data proxy rewards built from cross-fitted doubly robust pseudo-outcomes for IHDP, ACIC 2016, and normalized LaLonde, and from a DR difference-in-differences pseudo-target for

*Table 2.* **Representative solution families discovered by evolution.** Final programs cluster into dataset-specific estimator families with recognizable refinement patterns.

| Dataset | Dominant family | Typical refinements during evolution | Within-dataset TF-IDF | Closest official reference | Median closest-ref. emb. cosine |
|---|---|---|---|---|---|
| ACIC 2022 | TWFE / DiD / panel FE | Pre-period anchoring, detrending, weighting, clustered inference, calibration | $0.336 \pm 0.110$ | IPW | 0.585 |
| ACIC 2016 | Meta-learners + DR | Cross-fitting, clipping, covariate encoding, X-/DR learners, and AIPW aggregation | $0.353 \pm 0.173$ | DR-learner | 0.678 |
| IHDP | Boosted/forest T-/X-learners | Outcome tuning, DR pseudo-outcomes, ATE calibration, and stable nuisance fits | $0.415 \pm 0.244$ | Causal forest | 0.691 |
| LaLonde | Weighting/DR hybrids for observational bias | Propensity weighting, AIPW/DR correction, adaptive regularization, and small-sample ensembling | $0.331 \pm 0.116$ | DR-learner | 0.707 |

ACIC 2022 (see Methods). This lets the same evolutionary loop operate even when ground-truth counterfactuals are unavailable.

Proxy fidelity was benchmark-dependent but informative. On matched final runs, proxy-to-true combined score correlation was strongest on IHDP ($r = 0.97$), lower on normalized LaLonde ($r = 0.77$) and ACIC 2016 ($r = 0.50$), and for ACIC 2022 the DR-DID hit-rate component tracked true coverage very closely across checkpoint reevaluations ($r = 0.99$). The strongest proxy-guided estimators beat selected baselines on IHDP and ACIC 2022; on LaLonde, they improved over the strongest package heterogeneity baseline even though they no longer beat the stronger design-based ATE comparator under the 20:80 main split (Table 1). ACIC 2016 remained the clearest failure case, highlighting that optimal proxy design remains an open problem rather than a solved component.

## 4. Discussion, Conclusion, and Limitations

With the growing scientific and coding capabilities of language models, their use in causal inference offers a promising route for automating empirical causal analysis. We introduced `InferenceEvolve`, a knowledge- and feedback-guided evolutionary framework that searches causal-inference code spaces for empirically stronger estimators. Across widely used semi-synthetic benchmarks, `InferenceEvolve` produced a distribution of candidate estimators whose best and upper tail substantially improved over fixed baselines, while average gains varied by benchmark and objective. Under the prespecified 20:80 split, the strongest runs improved on the best baseline for all IHDP and ACIC 2016 metrics, most normalized LaLonde metrics, and several ACIC 2022 metrics; proxy-guided variants were competitive on some benchmarks but failed on others.

Our findings also show that proxy-guided estimator search is feasible but could be fragile: best proxy-guided variants improved on selected baselines in several benchmarks, but proxy fidelity and downstream performance were uneven. Moreover, as with all benchmark-driven causal-inference work, strong semi-synthetic performance does not guarantee success in real-world observational studies (Hill, 2011; Dorie et al., 2019; Thal & Finucane, 2023). Our search requires more computations than classical baselines, and the resulting programs can become specialized to benchmark-specific data-generating regimes. We therefore view `InferenceEvolve` as an automated estimator-search system for established benchmark regimes, not a replacement for expert causal analysis.

Beyond benchmark gains, our results suggest that LLM-guided evolutionary search can do more than tune predictors: it can adapt scientific procedures involving nuisance estimation, aggregation, uncertainty quantification, and implicit assumptions about data-generating regimes. In terms of broad impact, automated estimator search could make careful causal analysis more accessible and help stress-test methodological choices before deployment, but benchmark-optimized generated estimators could be over-trusted or misused to support causal claims without design checks, domain expertise, and important sensitivity analyses. We highlight the importance of human-AI collaboration here, especially for high-stakes applications in policy and health settings.

## Acknowledgements

We would like to thank the Causal Inference Working Group in the Department of Biostatistics at Johns Hopkins University and Alex Luedtke for helpful feedback. H.Z. and Y.C. are partially supported by the Data Science and AI Faculty Innovation Fund in the Department of Biostatistics at the Johns Hopkins Bloomberg School of Public Health.

## References

Acharya, S., Zhang, T. J., Kim, A., Haghighat, A., Sun, X., Shrestha, R. B., Mordig, M., Danisman, F., Jose, C., Qi, Y., Cobben, P., Schölkopf, B., Sachan, M., and Jin, Z. CauSciBench: Assessing LLM Causal Reasoning for Scientific Research. In *NeurIPS 2025 Workshop on CauScien: Uncovering Causality in Science*, October

2025. URL https://openreview.net/forum?id=EO8mTLqDuT.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Athey, S., Tibshirani, J., and Wager, S. Generalized random forests. *The Annals of Statistics*, 47(2):1148–1178, 2019. doi: 10.1214/18-AOS1709.

Austin, P. C. An introduction to propensity score methods for reducing the effects of confounding in observational studies. *Multivariate Behavioral Research*, 46(3):399–424, 2011.

Bailey, D. H., Jung, A. J., Beltz, A. M., Eronen, M. I., Gische, C., Hamaker, E. L., Kording, K. P., Lebel, C., Lindquist, M. A., Moeller, J., et al. Causal inference on human behaviour. *Nature Human Behaviour*, 8(8): 1448–1459, 2024.

Bang, H. and Robins, J. M. Doubly robust estimation in missing data and causal inference models. *Biometrics*, 61 (4):962–973, 2005.

Battocchi, K., Dillon, E., Hei, M., Lewis, G., Oka, P., Oprescu, M., and Syrgkanis, V. EconML: A Python Package for ML-Based Heterogeneous Treatment Effects Estimation. GitHub repository, 2019. URL https://github.com/py-why/EconML.

Berrevoets, J., Piskorz, J., Davis, R., Amad, H., Weatherall, J., and van der Schaar, M. Technical report: Facilitating the adoption of causal inference methods through LLM-empowered co-pilot. *arXiv preprint arXiv:2508.10581*, 2025. doi: 10.48550/arXiv.2508.10581. URL https://arxiv.org/abs/2508.10581.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., and Robins, J. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68, 2018. doi: 10.1111/ectj.12097.

Dehejia, R. H. and Wahba, S. Causal effects in nonexperimental studies: Reevaluating the evaluation of training programs. *Journal of the American Statistical Association*, 94(448):1053–1062, 1999.

Dhawan, N., Cotta, L., Ullrich, K., Krishnan, R. G., and Maddison, C. J. End-to-end causal effect estimation from unstructured natural language data. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/8d2aad53fa148791ffd004a7c9cce5ad-Abstract-Conference.html.

Dorie, V., Hill, J., Shalit, U., Scott, M., and Cervone, D. Automated versus do-it-yourself methods for causal inference: Lessons learned from a data analysis competition. *Statistical Science*, 34(1):43–68, 2019.

Feuerriegel, S., Frauen, D., Melnychuk, V., Schweisthal, J., Hess, K., Curth, A., Bauer, S., Kilbertus, N., Kohane, I. S., and van der Schaar, M. Causal machine learning for predicting treatment outcomes. *Nature Medicine*, 30(4): 958–968, 2024.

Gao, S., Fang, A., Huang, Y., Giunchiglia, V., Noori, A., Schwarz, J. R., Ektefaie, Y., Kondic, J., and Zitnik, M. Empowering biomedical discovery with AI agents. *Cell*, 187(22):6125–6151, 2024.

Gottweis, J., Weng, W., Daryin, A., Tu, T., Palepu, A., Sirkovic, P., and Natarajan, V. Towards an AI co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.

Hernán, M. A. and Robins, J. M. *Causal Inference: What If*. Chapman & Hall/CRC, 2020.

Hill, J. L. Bayesian Nonparametric Modeling for Causal Inference. *Journal of Computational and Graphical Statistics*, 20(1):217–240, January 2011. ISSN 1061-8600. doi: 10.1198/jcgs.2010.08162. URL https://doi.org/10.1198/jcgs.2010.08162. Publisher: ASA Website _eprint: https://doi.org/10.1198/jcgs.2010.08162.

Imbens, G. W. and Rubin, D. B. *Causal inference in statistics, social, and biomedical sciences*. Cambridge University Press, 2015.

Karpathy, A. autoresearch, 2026. URL https://github.com/karpathy/autoresearch. GitHub repository, accessed 2026-04-05.

Kennedy, E. H. Towards optimal doubly robust estimation of heterogeneous causal effects. *Electronic Journal of Statistics*, 17(2):3008–3049, 2023.

Kennedy, E. H. Semiparametric doubly robust targeted double machine learning: a review. *Handbook of Statistical Methods for Precision Medicine*, pp. 207–236, 2024.

Kraev, E., Flesch, T., Lekunze, H. T., Harley, M., and Planell Morell, P. Out-of-sample scoring and automatic selection of causal estimators. *arXiv preprint arXiv:2212.10076*, 2022. doi: 10.48550/arXiv.2212.10076. URL https://arxiv.org/abs/2212.10076.

Künzel, S. R., Sekhon, J. S., Bickel, P. J., and Yu, B. Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the National Academy of Sciences*, 116(10):4156–4165, 2019.

LaLonde, R. J. Evaluating the econometric evaluations of training programs with experimental data. *The American Economic Review*, 76(4):604–620, 1986.

Mahajan, D., Mitliagkas, I., Neal, B., and Syrgkanis, V. Empirical analysis of model selection for heterogeneous causal effect estimation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=yuy6cGt3KL.

Miettinen, K. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 1999.

Mouret, J.-B. and Clune, J. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Nguyen, H., Grover, P., and Khatwani, D. Opportunityfinder: A framework for automated causal inference, 2023. URL https://arxiv.org/abs/2309.13103.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J., Mehrabian, A., et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Pearl, J. Causal inference in statistics: An overview. *Statistics Surveys*, 3:96–146, 2009.

Rosenbaum, P. R. and Rubin, D. B. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.

Sharma, A. Openevolve: an open-source evolutionary coding agent. https://github.com/codelion/openevolve, 2025.

Stuart, E. A. Matching methods for causal inference: A review and a look forward. *Statistical Science*, 25(1):1, 2010.

Swanson, K., Wu, W., Bulaong, N. L., Pak, J. E., and Zou, J. The virtual lab of AI agents designs new sars-cov-2 nanobodies. *Nature*, 646(8085):716–723, 2025. doi: 10.1038/s41586-025-09442-9.

Team, G., Georgiev, P., Lei, V. I., Burnell, R., Bai, L., Gulati, A., Tanzer, G., Vincent, D., Pan, Z., Wang, S., et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Thal, D. R. and Finucane, M. M. Causal Methods Madness: Lessons Learned from the 2022 ACIC Competition to Estimate Health Policy Impacts. *Observational Studies*, 9(3):3–27, 2023. ISSN 2767-3324. URL https://muse.jhu.edu/pub/56/article/895650. Publisher: University of Pennsylvania Press.

Vanderschueren, T., Verdonck, T., van der Schaar, M., and Verbeke, W. AutoCATE: End-to-end, automated treatment effect estimation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 60880–60904. PMLR, 2025. URL https://proceedings.mlr.press/v267/vanderschueren25a.html.

Verma, V., Acharya, S., Bhardwaj, D., Simko, S., Yang, Y., Haghighat, A., Janzing, D., Sachan, M., Schölkopf, B., and Jin, Z. Causal AI scientist: Facilitating causal data science with large language models. In *NeurIPS 2025 Workshop on CauScien: Uncovering Causality in Science*, 2025. URL https://openreview.net/forum?id=EDWTHMVOCj.

Wager, S. and Athey, S. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 113(523):1228–1242, 2018.

Wang, Z., Liu, F., Feng, Q., Zhang, Q., and Gao, X. LLM-enhanced score function evolution for causal structure learning. In Kwok, J. (ed.), *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 9086–9094. International Joint Conferences on Artificial Intelligence Organization, 8 2025. doi: 10.24963/ijcai.2025/1010. URL https://doi.org/10.24963/ijcai.2025/1010. Main Track.

Yamada, Y., Lange, R. T., Lu, C., Hu, S., Lu, C., Foerster, J., Clune, J., and Ha, D. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

## Data availability

The benchmark datasets used in this study are publicly available from their original sources: ACIC 2022 (https://acic2022.mathematica.org/data-challenge/), ACIC 2016 (https://github.com/vdorie/aciccomp), the IHDP benchmark replications (https://github.com/AMLab-Amsterdam/CEVAE), and the Dehejia–Wahba / LaLonde data distributed through CRAN packages such as `Matching` (https://search.r-project.org/CRAN/refmans/Matching/html/lalonde.html). Exact download instructions, preprocessing steps, and benchmark-specific wrappers are documented in the project repository. The release documentation will list source licenses and terms of use for each dataset and package, including Creative Commons Attribution 4.0 (CC BY 4.0) where indicated by benchmark-data providers and package licenses such as GPL-3 for CRAN assets. An interactive exploration of the benchmark results and evolved programs is available at https://yiqunchen.github.io/causal-agent/.

## Code availability

Code for `InferenceEvolve`, including prompts, evaluation scripts, figure generation, and analysis workflows, is available at https://github.com/yiqunchen/causal-agent. The repository also documents dataset access, preprocessing, per-run configurations, asset licenses and terms of use, and the final evolved programs used in this manuscript. The de-anonymized code release is intended to use CC BY 4.0, or the closest compatible open license permitted by repository dependencies.

# A. Supplementary Material

This supplementary section is organized to parallel the main text. We first document the exact prompts, search workflow, and scoring rules that define the estimator-discovery problem. We then collect the additional benchmark summaries and sensitivity analyses referenced from the Results section, before turning to dataset-specific evolutionary trajectories and representative code edits.

## A.1. S1. Search setup and prompts

This section defines the estimator-search problem solved by `InferenceEvolve`. We begin with the exact prompt templates supplied to the LLMs, then summarize the common workflow, the fixed configuration choices, the benchmark-specific scoring rules, and the implementation details needed to reproduce the experiment.

### A.1.1. S1.1. Evolution prompts

The four prompt templates below were used to initialize the evolutionary search across benchmarks. Each prompt fixes the data schema, return signature, evaluation metric, allowed libraries, and mutation rules; a short data preview was appended dynamically at runtime.

## ACIC 2022 Evolution Prompt

```
You are an expert causal inference statistician and Python programmer.

Background:
- We have multisite randomized trial data in long format (practice-year level).
- Treatment is assigned at the practice level, randomized, and 'post' marks the
  period after treatment rollout.

Data columns:
   - 'id.practice' (int): site identifier
   - 'year' (int): panel time index
   - 'Y' (float): average monthly spending per patient
   - 'Z' (0/1): treatment indicator
   - 'post' (0/1): post-treatment period indicator
   - 'n.patients' (int): patient count (weights)
   - Practice-level covariates: X1..X9
   - Aggregated patient covariates: V1_avg..V5_C_avg

Task: Estimate the Overall SATT for treated practices in the post-treatment
period, weighted by n.patients.

OpenEvolve Objective:
- Coverage gap is |coverage - 0.9|/0.9.
- Score = log(1+RMSE) + lambda * log(1+coverage_gap).
- Combined score reported to OpenEvolve is 1/(1+score).
- Minimise score by reducing RMSE while achieving close to 90% coverage.

Function signature:
   def estimate(df) -> (tau_hat, lb, ub, method)

Constraints: Pure, deterministic. Use only pandas, numpy, scikit-learn,
statsmodels. Robust to NaNs and edge cases. Modify only the EVOLVE-BLOCK.

[Data preview appended at runtime]
```

## IHDP Evolution Prompt

```
You are an expert causal inference researcher and Python programmer working on
the IHDP semi-synthetic benchmark.

Dataset schema: `treatment` (binary), `y_factual` (outcome), `x1`..`x25`
(covariates). Ground-truth columns removed before execution.

Task: Implement a deterministic estimator returning (ite_hat, ate_hat,
method_name).

Evaluation: Score = log(1+mean_sqrt_PEHE) + lambda*log(1+mean_ATE_err).
The combined score reported to OpenEvolve is 1/(1+score).
50 replicates; configurable train/test split.

Objective: Minimise score. Explore diverse estimator families (T-/X-/DR-learners,
doubly-robust, etc.). Prioritise stability.

Requirements: Pure function. pandas, numpy, scikit-learn, statsmodels only.
Modify only the EVOLVE-BLOCK.

[Data preview appended at runtime]
```

## ACIC 2016 Evolution Prompt

```
You are an expert causal inference researcher and Python programmer working on
the ACIC 2016 semi-synthetic benchmark.

Dataset schema: `treatment` (binary), `y_factual` (outcome), `x_1`..`x_58`
(covariates, mix of continuous and categorical). Ground-truth columns removed.

Task: Implement a deterministic estimator returning (ite_hat, ate_hat,
method_name).

Evaluation: source pool of 200 replicates (5 DGPs x 40 simulations);
main study uses a 100-replicate working subset per run.
Score = log(1+mean_sqrt_PEHE) + lambda*log(1+mean_ATE_err).
The combined score reported to OpenEvolve is 1/(1+score).

Objective: Minimise score. Explore diverse estimator families. Handle non-numeric
covariates robustly.

Requirements: Pure function. pandas, numpy, scikit-learn, statsmodels only.
Modify only the EVOLVE-BLOCK.

[Data preview appended at runtime]
```

## LaLonde Evolution Prompt

```
You are an expert causal inference researcher and Python programmer working on
the LaLonde observational dataset (Dehejia-Wahba / PSID controls).

Dataset schema: `treatment` (binary), `y_factual` (re78 earnings, $0-$107K),
covariates: age, education, black, hispanic, married, nodegree, re74, re75.
Ground-truth columns y0, y1, and ite exist in the benchmark files but are
removed before candidate execution.

Domain guidance:
- Observational study with strong selection bias between groups.
- Prior earnings (re74, re75) are strongest predictors.
- Consider propensity-score trimming and overlap enforcement.
- Do not assume ATE is positive.

Task: Implement a deterministic estimator returning (ite_hat, ate_hat,
method_name).

Evaluation: 100 semi-synthetic replicates.
Score = log(1+normalized_sqrt_PEHE) + lambda*log(1+normalized_ATE_err).
The combined score reported to OpenEvolve is 1/(1+score).

Requirements: Pure function. pandas, numpy, scikit-learn, statsmodels only.
Modify only the EVOLVE-BLOCK.

[Data preview appended at runtime]
```

### A.1.2. S1.2. ALGORITHMIC WORKFLOW

Algorithm 1 in the main text summarizes the common search loop used across all 96 runs in the factorial experiment; the fixed hyperparameters, dataset splits, and scoring rules are detailed below.

**Iterative mutation prompts.** After the seed program, OpenEvolve used its built-in full-rewrite mutation template. Each prompt included the current parent program, current metrics and feature coordinates, previous attempts, top archived programs, sampled inspiration programs, and the benchmark-specific task constraints. The model returned a complete rewritten program with the same `estimate(...)` interface; complete prompt logs are archived with the evolution traces.

### A.1.3. S1.3. FULL HYPERPARAMETER CONFIGURATION

Supplementary Table 1 lists the fixed settings and factorial factors used throughout the full benchmark sweep.

*Table 1.* **Complete hyperparameter configuration.** Parameters held constant across the 96-run factorial experiment unless noted. Factorial factors indicated with †.

| Parameter | Value |
|---|---|
| *Evolutionary search* | |
| Maximum iterations | 100 |
| Population size (MAP-Elites) | 60 |
| Number of islands | 4 |
| Archive size per island | 25 |
| Elite selection ratio | 0.3 |
| Exploitation ratio | 0.7 |
| Allow full rewrites | True |
| Diff-based evolution | False |
| Number of top programs in prompt | 3 |
| Checkpoint interval | 5 iterations |
| *LLM configuration* | |
| API provider | OpenRouter |
| Primary model† | GPT-5 or Gemini 3 Flash Preview |
| Primary model weight | 0.80 |
| Secondary model | Claude Sonnet 4 |
| Secondary model weight | 0.20 |
| Temperature | 0.7 |
| Top-$p$ | 0.95 |
| Max output tokens | 8,192 |
| API timeout | 600 s |
| *Evaluation pipeline* | |
| Cascade evaluation | True |
| Stage 1 threshold ($\theta_1$) | 0.001 |
| Parallel workers (stage 2 search) | 1 |
| Parallel workers (final held-out evaluation) | 4 |
| Evaluation timeout per program | 1,800 s |
| *Dataset-specific* | |
| ACIC 2022: training / held-out replicates | 75 / 75 (50:50) or 30 / 120 (20:80), from a 150-replicate working subset sampled from 300 total |
| ACIC 2022: target coverage | 0.90 |
| IHDP: training / held-out replicates | 12 / 13 (50:50) or 5 / 20 (20:80), from a 25-replicate working subset sampled from 50 total |
| ACIC 2016: training / held-out replicates | 50 / 50 (50:50) or 20 / 80 (20:80), from a 100-replicate working subset sampled from 200 total |
| LaLonde: training / held-out replicates | 25 / 25 (50:50) or 10 / 40 (20:80), from a 50-replicate working subset sampled from 100 total |
| *Factorial factors*† | |
| Datasets | ACIC 2022, ACIC 2016, IHDP, LaLonde |
| LLM (primary) | GPT-5, Gemini 3 Flash Preview |
| Regularization $\lambda$ | 0.2 (small), 1.0 (moderate), 5.0 (large) |
| Train–test split | 50:50, 20:80 |
| Independent replicates per cell | 2 |
| *Reproducibility* | |
| Deterministic replicate seeds | 20250215, 20250216 |
| Allowed Python packages | pandas, numpy, scikit-learn, statsmodels |

#### A.1.4. S1.4. Composite scoring functions

Each benchmark uses a scalarized score so that MAP-Elites selection can trade off primary accuracy against secondary objectives such as ATE calibration or interval coverage.

**ACIC 2022 (panel data):** $S_{\text{ACIC}} = 1/(1 + \log(1 + \text{RMSE}) + \lambda \cdot \log(1 + |\text{Coverage} - 0.9|/0.9))$.

**IHDP, ACIC 2016, LaLonde (cross-sectional):** $S_{\text{ITE}} = 1/(1 + \log(1 + \overline{\sqrt{\text{PEHE}}}) + \lambda \cdot \log(1 + \overline{|\text{ATE error}|}))$, where the overlines denote averages across the training replicates evaluated in each iteration and $\sqrt{\text{PEHE}}$ denotes the unit-level ITE RMSE from equation 1. The three levels of $\lambda$—0.2 (small), 1.0 (moderate), and 5.0 (large)—control the relative weight of secondary metrics.

#### A.1.5. S1.5. Implementation and reproducibility details

The notes below summarize the runtime environment and randomization controls used for the full experiment.

**Software.** The framework is implemented in Python 3.11 using OpenEvolve (Sharma, 2025) v0.2. LLM API calls are routed through OpenRouter. Candidate programs are executed in isolated subprocesses with enforced timeouts.

**MAP-Elites archive.** Each island used archive size 25 and population size 60. Bins were defined by OpenEvolve's built-in `complexity` and `diversity` feature dimensions, using 10 bins per dimension and a diversity reference set of 20 programs. Complexity is code length, and diversity is OpenEvolve's fast code-diversity score against the reference set.

**Reproducibility.** Two deterministic replicate seeds (20250215 and 20250216) control benchmark replicate sampling across the factorial grid. Per-run configuration files and evolved programs at every checkpoint are archived in the code repository.

**LLM call counts and stylized API cost.** Each evolution iteration issues one candidate-generation LLM call, so a 100-iteration run corresponds to 100 generation calls. Under the 0.80/0.20 primary/secondary ensemble weights, the expected composition is 80 primary-model calls and 20 Claude Sonnet 4 calls per run. The 96-run true-score factorial therefore corresponds to about 9,600 generation calls in total (expected 7,680 primary and 1,920 Claude calls); including the 48 matched proxy runs raises the total to about 14,400 calls. If a run uses on average $b_{\text{in}}$ prompt tokens and $b_{\text{out}}$ completion tokens per generation call, then its expected token budget is $100\, b_{\text{in}}$ input tokens and $100\, b_{\text{out}}$ output tokens, and the corresponding API spend is obtained by multiplying those totals by the provider's then-current model-specific per-token prices. For a representative bookkeeping example of 12k input and 2k output tokens per call, this corresponds to about 1.2M input tokens and 0.2M output tokens per run, about 115M/19M tokens over the 96-run main factorial, and about 173M/29M tokens when the 48 proxy runs are included.

### A.2. S2. Additional benchmark summaries and complexity analyses

This section collects the supporting tables and figures referenced from the Results section. We first report the full zero-shot benchmark comparisons, then document the exact program-analysis inputs and prompts, summarize code similarity across final evolved programs, and finally show the expanded code-length and methods-length figures that sit behind the main-text complexity discussion.

#### A.2.1. S2.1. Full zero-shot benchmark tables

Supplementary Tables 2 and 3 provide the complete zero-shot comparisons referenced in the ACIC 2022 and IHDP results discussion. They contextualize the seed programs from which the evolutionary search began.

*Table 2.* **Full zero-shot results on ACIC 2022 (50 replicates).** Seven LLMs evaluated under identical conditions. Each model was given the same structured prompt containing the data schema, causal assumptions, evaluation criteria, and a six-row data preview. Models were allowed up to five attempts to produce a valid program passing a 180 s warm-up test on a single replicate.

| Model | Attempts | RMSE | Coverage | Method |
|---|---|---|---|---|
| Claude Sonnet 4 | 2 | 89.24 | 0.12 | Weighted AIPW w/ RF |
| GPT-5 | 1 | 33.05 | 0.40 | Patient-weighted TWFE DiD |
| GPT-4o | 1 | 39.21 | 0.36 | DiD + regression adjustment |
| Grok 4 Fast | 3 | 89.24 | 0.22 | Diff-in-means w/ bootstrap |
| Qwen3-Coder | 1 | 68.93 | 0.30 | Weighted TWFE (clustered) |
| DeepSeek v3.1 | 2 | 19.44 | 0.02 | Weighted FE w/ clustered SEs |
| Gemini 2.5 Pro | 5 | — | — | *No valid program* |

*Table 3.* **Full zero-shot results on IHDP (50 replicates).** Seven LLMs evaluated under identical conditions. Means $\pm$ SE are computed across 50 replicates with ground-truth potential outcomes.

| Model | Attempts | $\sqrt{\text{PEHE}}$ | $\lvert\text{ATE error}\rvert$ |
|---|---|---|---|
| Claude Sonnet 4 | 1 | $2.40 \pm 0.55$ | $0.103 \pm 0.013$ |
| GPT-5 | 1 | $5.68 \pm 0.73$ | $0.292 \pm 0.030$ |
| GPT-4o | 1 | $1.86 \pm 0.38$ | $0.104 \pm 0.014$ |
| Grok 4 Fast | 1 | $1.86 \pm 0.38$ | $0.106 \pm 0.014$ |
| Qwen3-Coder | 2 | $3.11 \pm 0.73$ | $0.489 \pm 0.141$ |
| DeepSeek v3.1 | 2 | $2.02 \pm 0.34$ | $0.103 \pm 0.018$ |
| Gemini 2.5 Pro | 5 | — | — |

### A.2.2. S2.2. Program-analysis inputs and prompt summaries

This subsection documents the code pool and prompt templates behind the program-analysis results in Fig. 3, Table 2, and the complexity figures below. We analyzed one final code snippet per run: 28 zero-shot baselines, 96 true-evolved finals across both split regimes, and 48 matched proxy-evolved finals from the 20:80 proxy analysis. All embedding-based figures and tables in the main text use OpenAI `text-embedding-3-large`. We reran the same within- versus between-dataset and nearest-reference summaries with `Qwen3-Embedding-0.6B` as a sensitivity analysis; the alternative embedding space was more compressed, but it preserved the same qualitative ordering and dataset-level conclusions.

For the official reference library, we also archived the exact commit-pinned upstream example links used to build the seven normalized wrappers (see Supplementary Table 5).

*Table 4.* **Program-analysis inputs and prompt summaries.** Exact code sources and shared prompt templates for the program-analysis pipeline.

| Component | Models / inputs | Retrieval rule or shared prompt |
|---|---|---|
| Program pool | 28 baselines, 96 true-evolved finals, 48 proxy-evolved finals | One final snippet per run from the zero-shot baseline pool, the true-score final-program pool, and the proxy-score final-program pool. The exact 172-program manifest is archived alongside the analysis outputs. |
| Translation | GPT-5, Gemini 2.5 Pro, Claude Sonnet 4.5 | Shared system and user template shown below; each prompt supplied the dataset name, source label, and full program text, and asked for a single manuscript-style methods paragraph. |
| Judge panel | GPT-5, Gemini 2.5 Pro, Claude Sonnet 4.5 | Shared JSON-returning review template shown below; each prompt asked for family assignment, nearest known method, complexity, pipeline steps, and novel components before cross-model aggregation. |
| Embedding sensitivity | OpenAI main; Qwen sensitivity | All manuscript figures and tables use `text-embedding-3-large`. The same nearest-reference and within/between-program summaries were rerun with `Qwen3-Embedding-0.6B` only as a sensitivity analysis. |

*Table 5.* **Commit-pinned upstream sources for the official reference library.** Each normalized `estimate(df)` wrapper used in the reference-matching analysis was adapted from the corresponding maintainer example at the listed commit-pinned URL.

| Family | Wrapper | Package | Commit-pinned upstream example | Commit |
|---|---|---|---|---|
| T-learner | T-Learner (EconML TLearner) | EconML | https://github.com/py-why/EconML/blob/b0404c02be5af1f9a2cbc2bd66d9b47e38c6ebb4/econml/metalearners/_metalearners.py#L54-L62 | b0404c0 |
| X-learner | X-Learner (EconML XLearner) | EconML | https://github.com/py-why/EconML/blob/b0404c02be5af1f9a2cbc2bd66d9b47e econml/metalearners/_metalearners.py#L328-L336 | b0404c0 |
| DR-learner | DR-Learner (EconML DR-Learner) | EconML | https://github.com/py-why/EconML/blob/b0404c02be5af1f9a2cbc2bd66d9b47e38c6ebb4/econml/dr/_drlearner.py#L542-L555 | b0404c0 |
| IPW | IPW (zEpid IPTW) | zEpid | https://github.com/pzivich/zEpid/blob/16a0f96f8b2c65df8715085801f21757 zepid/causal/ipw/IPTW.py#L85-L88 | 16a0f96 |
| OLS | OLS (statsmodels OLS) | statsmodels | https://github.com/statsmodels/statsmodels/blob/184e7f8e5725613ba64cbabc9bf0a33354e1d95e/examples/python/ols.py#L31-L37 | 184e7f8 |
| AIPW | AIPW (zEpid AIPTW) | zEpid | https://github.com/pzivich/zEpid/blob/16a0f96f8b2c65df8715085801f21757 zepid/causal/doublyrobust/AIPW.py#L74-L77 | 16a0f96 |
| Causal forest | CausalForest (EconML CausalForestDML) | EconML | https://github.com/py-why/EconML/blob/b0404c02be5af1f9a2cbc2bd66d9b47e38c6ebb4/econml/dml/causal_forest.py#L551-L558 | b0404c0 |

## Code-to-Methods Translation Prompt

```
System:
You convert Python estimation code into a methods paragraph for a scientific
manuscript. Describe the statistical procedure, preprocessing, weighting,
model structure, effect-estimation target, and uncertainty quantification when
present. Describe only what the code actually implements. Do not speculate, do
not mention absent steps, and do not add commentary such as "the code does not
explicitly..." or "there is no..." unless the omission is itself an explicit
design choice encoded in the program. Avoid line-by-line narration, code
syntax, bullet points, or implementation trivia that would not belong in a
methods section. Return plain text only.

User:
Dataset: <dataset>
Program source: <source label>

Translate the following Python code into a single publication-style methods
paragraph.

```python
<full program inserted here>
```
```

## LLM-as-a-Judge Aggregation Prompt

```
You are a causal inference expert reviewing a Python program that estimates
individual treatment effects (ITE) and average treatment effects (ATE).

Analyze this program and answer the following questions. Be precise and concise.

```python
<full program inserted here>
```

Respond in EXACTLY this JSON format (no markdown, no extra text):
{
  "algorithm_family": "<primary method family: one of T-Learner, S-Learner,
    X-Learner, DR-Learner, AIPW, IPW, OLS, CausalForest, Matching, Ensemble,
    Other>",
  "sub_methods": ["<specific sub-methods used>"],
  "most_similar_known_method": "<single most similar published method>",
  "similarity_to_known": <integer 1-10>,
  "novel_components": ["<algorithmic innovations not in the standard method>"],
  "novelty_explanation": "<1-2 sentence explanation>",
  "base_learners": ["<ML models used>"],
  "pipeline_steps": ["<ordered algorithm steps>"],
  "complexity_rating": <integer 1-10>
}
```

### A.2.3. S2.3. Code similarity among final programs

Supplementary Table 6 complements the main-text method-family summary by quantifying within- and between-dataset lexical similarity across the final evolved programs.

*Table 6.* **Code similarity among final programs.** TF-IDF cosine similarity between pairs of final evolved programs. Within-dataset similarity is consistently higher than between-dataset similarity, confirming genuine dataset-specific adaptation.

| Comparison | Mean $\pm$ SD | $n$ pairs |
|---|---|---|
| Within ACIC 2022 | $0.336 \pm 0.110$ | 276 |
| Within ACIC 2016 | $0.353 \pm 0.173$ | 276 |
| Within IHDP | $0.415 \pm 0.244$ | 276 |
| Within LaLonde | $0.331 \pm 0.116$ | 276 |
| ACIC 2022 vs. IHDP | $0.140 \pm 0.053$ | 576 |
| ACIC 2016 vs. IHDP | $0.352 \pm 0.186$ | 576 |
| ACIC 2022 vs. LaLonde | $0.148 \pm 0.063$ | 576 |

### A.2.4. S2.4. Program and methods-length summaries

Supplementary Figs. 1, 2, 3, and 4 expand the compact complexity panels in Fig. 2. They show the full distributions of source-code length and manuscript-style methods length across datasets, translator models, and evolutionary settings.

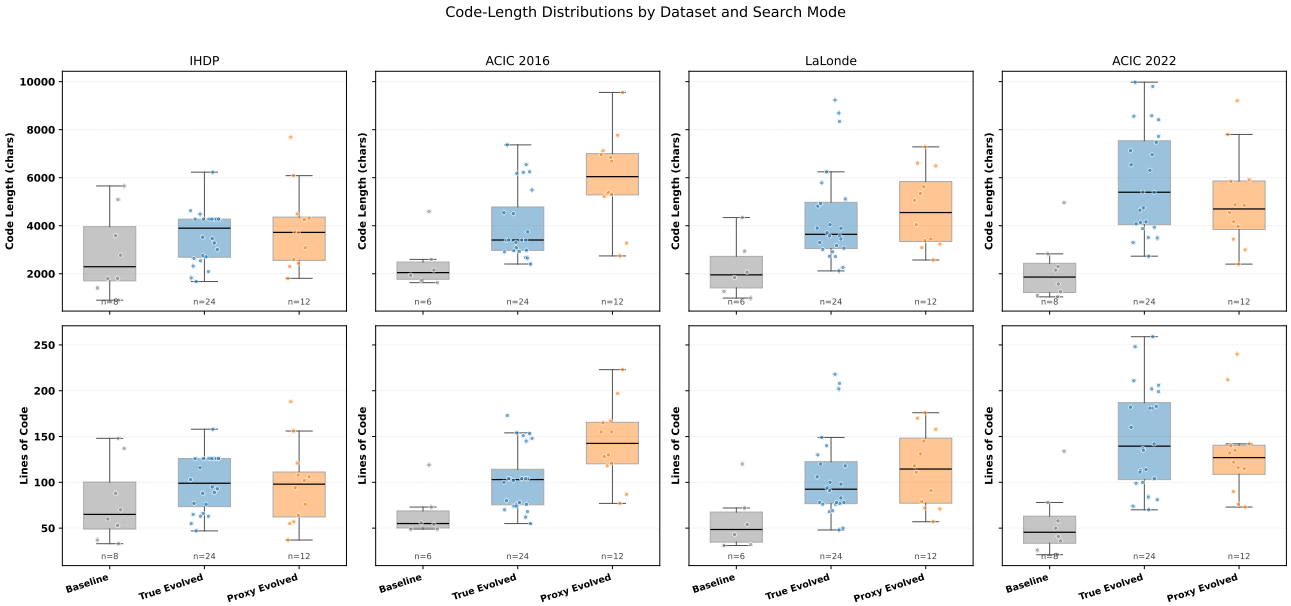


*Figure 1.* **Code length distributions across datasets and program sources.** Top panels show character count and bottom panels show non-empty lines of code for baseline, true-evolved, and proxy-evolved programs. Across all four benchmarks, evolved programs are consistently longer than the compact baseline programs, with the largest expansions occurring for ACIC 2022 true-evolved code and ACIC 2016 proxy-evolved code.

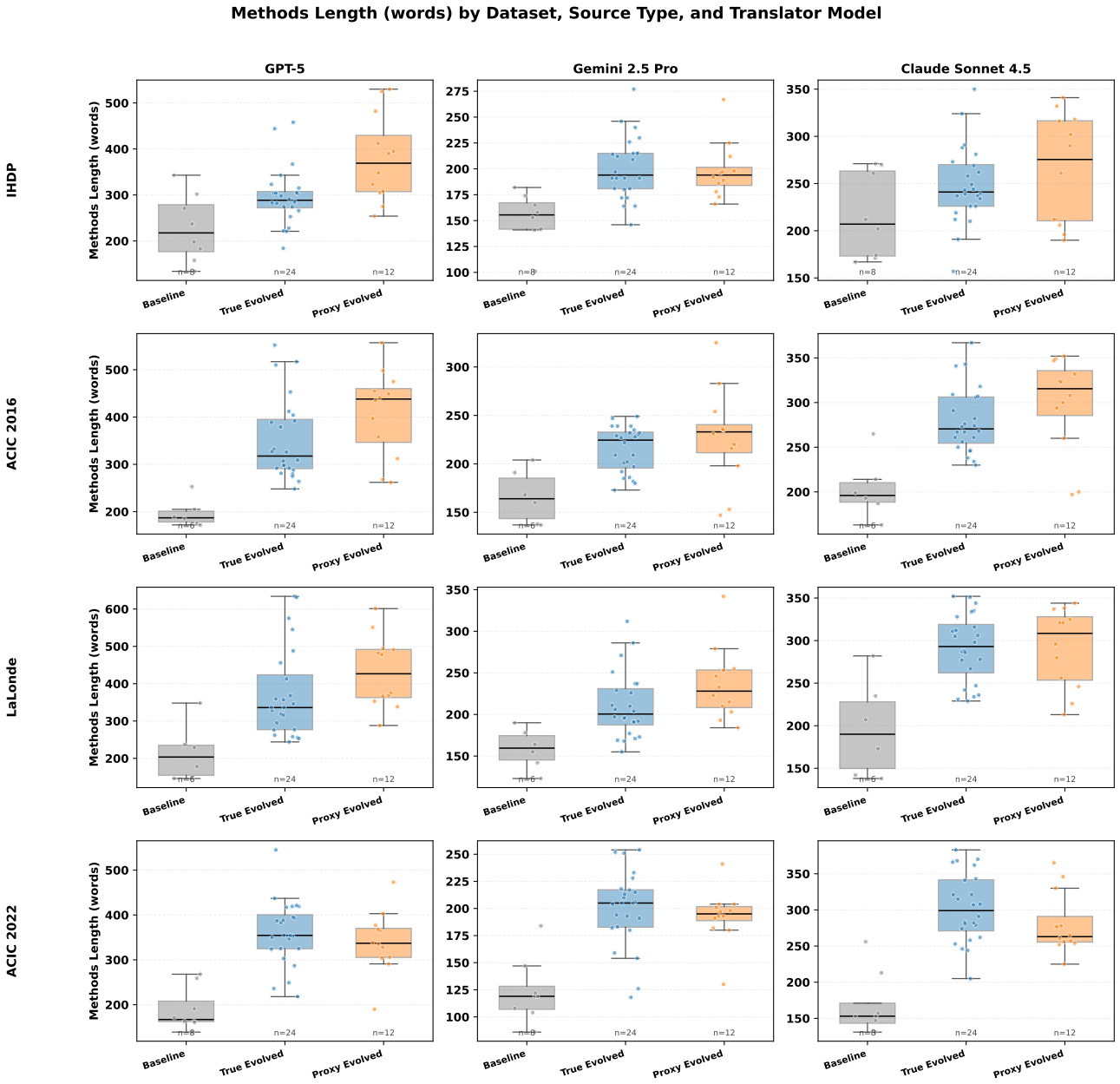


*Figure 2.* **Lengths of manuscript-style methods paragraphs derived from code.** Each program was translated into a single scientific-manuscript methods paragraph by GPT-5, Gemini 2.5 Pro, and Claude Sonnet 4.5 using the same unconstrained instruction template. Across datasets and translator models, baseline programs yield the shortest descriptions, while true-evolved and proxy-evolved programs produce substantially longer methods paragraphs.

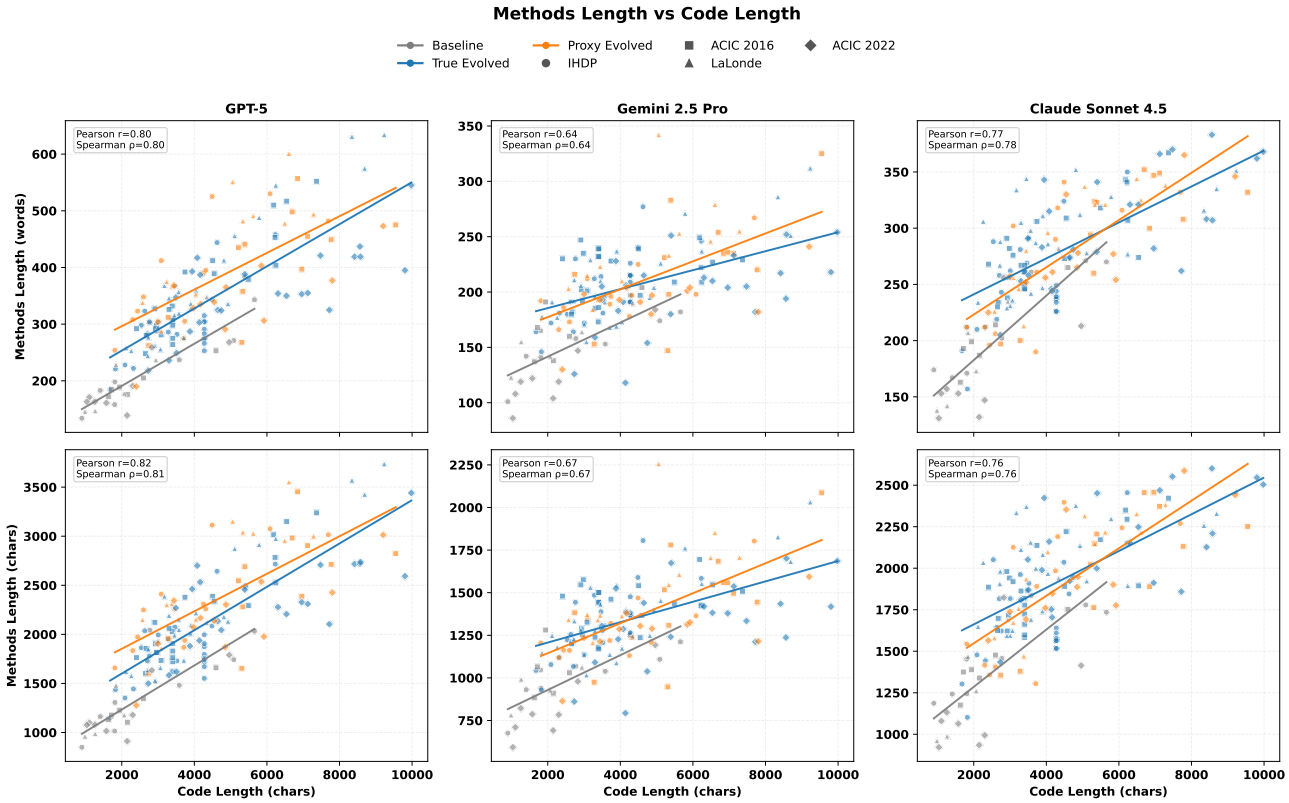


*Figure 3.* **Translated methods length scales with source-code length.** Each point is one code-to-method translation. Longer source programs tend to yield longer manuscript descriptions across all three translator models, indicating that structural complexity discovered during evolution is reflected in both code size and textual methodological detail.

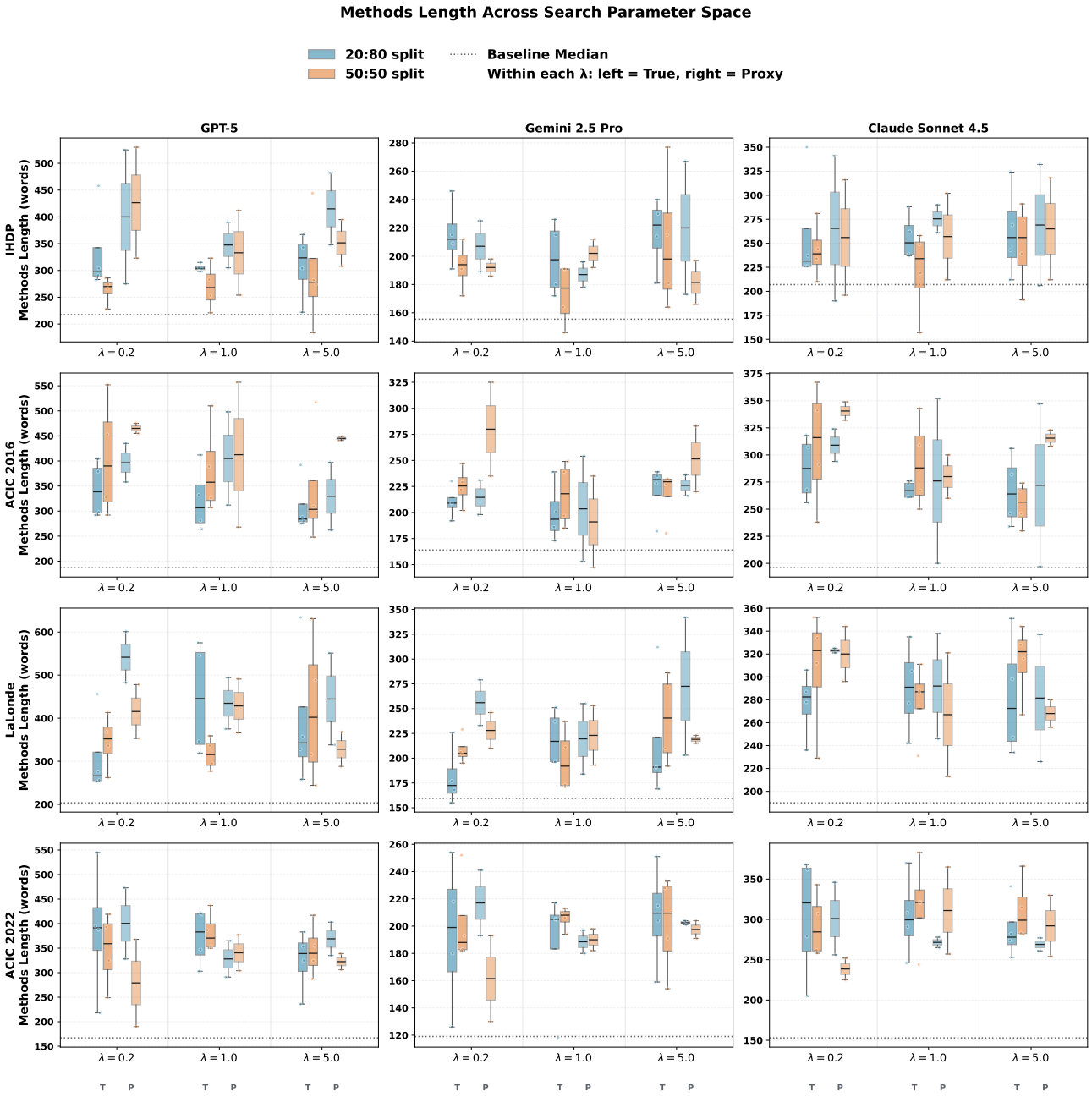


*Figure 4.* **Methods-paragraph length across the evolutionary parameter space.** Length distributions are stratified by dataset, translator model, source class, train–test split, and regularization level. Within each $\lambda$ group, the left pair shows true-evolved programs and the right pair shows proxy-evolved programs; box color denotes the 20:80 versus 50:50 split. Source effects dominate split and regularization effects, although GPT-5 proxy-derived descriptions are longest under small $\lambda$.

## A.3. S3. Sensitivity and proxy evaluation details

This section collects the supporting analyses behind the design-sensitivity and proxy-evaluation discussion in the Results section. We first document the held-out split sensitivity that is intentionally kept out of the main text, then decompose the regularization-weight comparison into its component metrics, and finally give the technical details and validation summaries for the proxy objectives.

### A.3.1. S3.1. Held-out split sensitivity

All main-text figures and tables use the prespecified 20:80 split. Supplementary Fig. 5 shows the matched 50:50 reruns for true-score evolution, pooling over regularization settings and replicate seeds within each split–model cell. Across datasets, the held-out metric distributions are similar under 20:80 and 50:50, with split-to-split shifts smaller than the benchmark-level differences emphasized in the main text. We therefore use 20:80 as the single anchor split in the Results section and treat 50:50 as a robustness check.

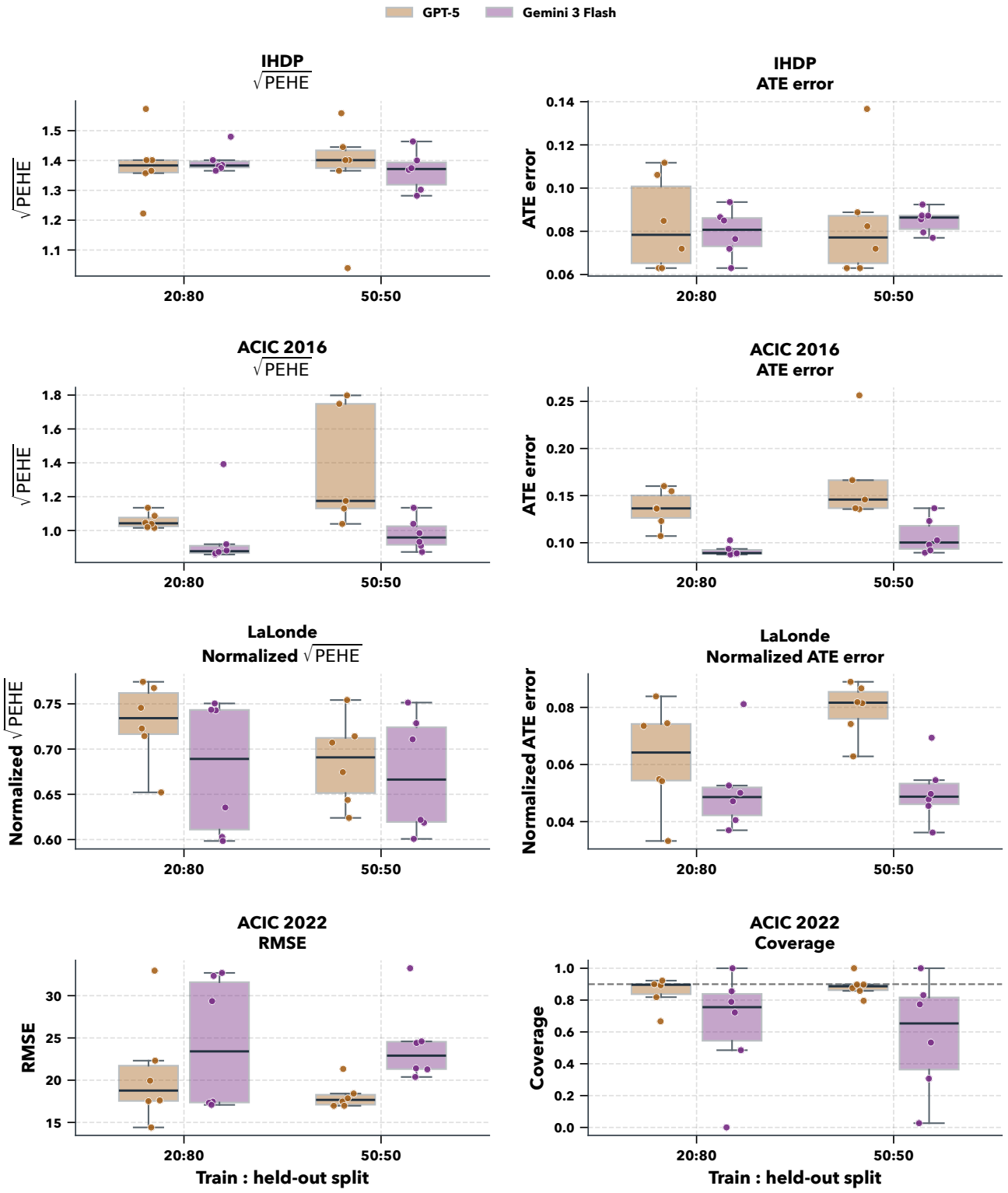


*Figure 5.* **Held-out sensitivity to the training split.** Each panel compares the prespecified 20:80 main-analysis runs with matched 50:50 reruns, pooling over regularization settings and replicate seeds within each split–model cell; box color denotes the primary model. Columns show the held-out metrics reported in the main text and tables: $\sqrt{\text{PEHE}}$ and ATE error for IHDP and ACIC 2016, normalized $\sqrt{\text{PEHE}}$ and normalized ATE error for LaLonde, and RMSE plus empirical coverage for ACIC 2022. The dashed line in the ACIC 2022 coverage panel marks nominal 90% coverage. The similar distributions across split settings motivate anchoring the main text on 20:80 alone.

### A.3.2. S3.2. Search dynamics and sensitivity to the regularization weight $\lambda$

Supplementary Fig. 6 separates the two primary models, GPT-5 and Gemini 3 Flash Preview, and shows that the effect of $\lambda$ is broadly similar across them. Smaller $\lambda$ generally favored lower held-out $\sqrt{\text{PEHE}}$ and/or ATE error on IHDP, ACIC 2016, and normalized LaLonde, whereas ACIC 2022 was comparatively flat over the tested range.

Supplementary Fig. 7 expands the final-score panel of Supplementary Fig. 6 into the combined score and the two component metrics used in each benchmark-specific objective, again under the main 20:80 split. Because $\lambda$ enters the scalarized score definition, only the component-metric columns support direct cross-$\lambda$ comparison of estimator quality.

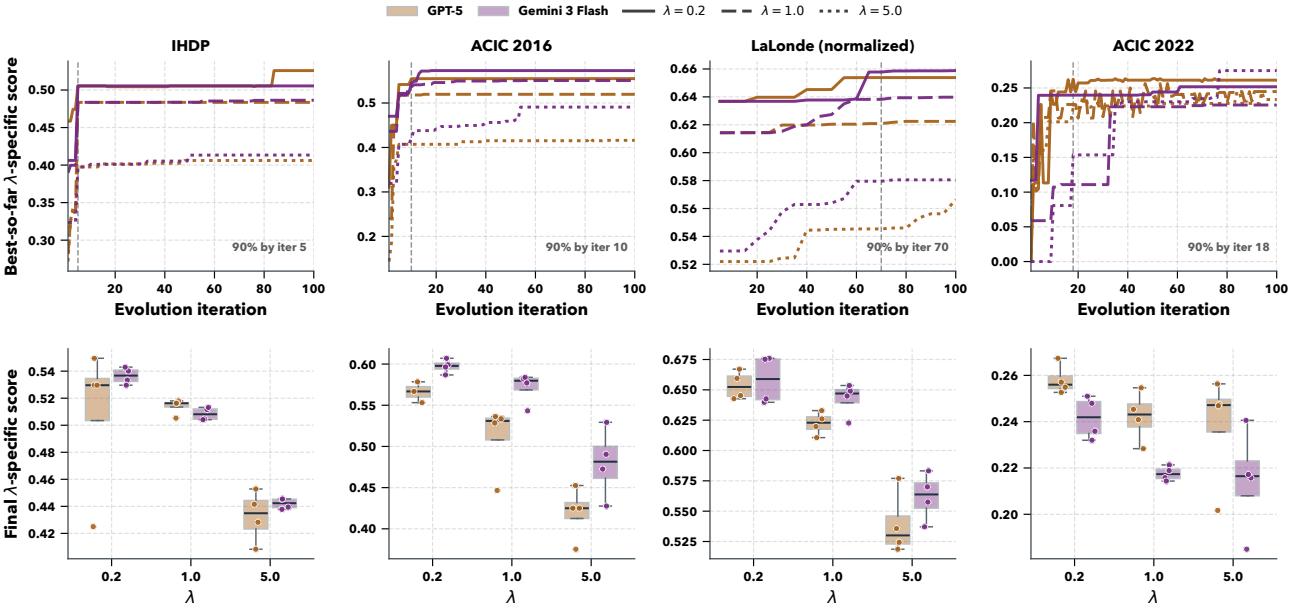


*Figure 6.* **Search dynamics across models and $\lambda$.** Top: median best-so-far scores over 100 iterations for 20:80 runs, stratified by model and $\lambda$; dashed lines mark the median iteration reaching 90% of the final best-so-far score. Bottom: final search-score distributions, comparable within but not across $\lambda$ settings.

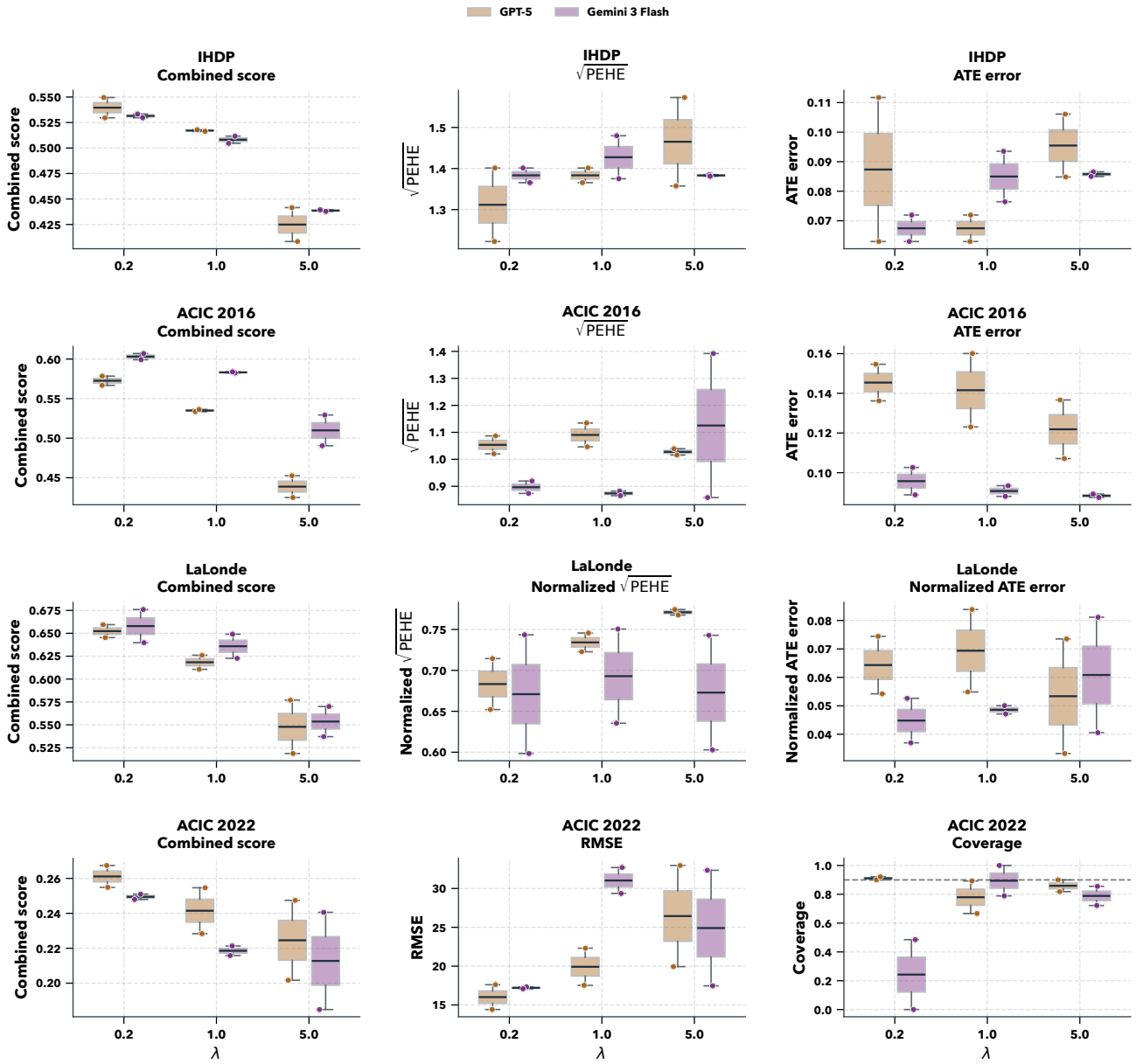


*Figure 7.* **Sensitivity to $\lambda$ across the combined score and its component metrics.** Each panel pools over replicate seeds within a dataset, $\lambda$, and primary-model cell under the main 20:80 split, with box colors denoting the primary model. Columns show the combined score, the primary accuracy metric, and the secondary metric used in the scalarized objective. Because $\lambda$ appears in the combined-score definition, the first column should be interpreted as a $\lambda$-specific search objective rather than a directly comparable cross-$\lambda$ performance scale; the second and third columns provide the valid held-out metric comparison. For IHDP, ACIC 2016, and LaLonde these component metrics are $\sqrt{\text{PEHE}}$ and ATE error (normalized for LaLonde); for ACIC 2022 they are RMSE and empirical coverage. The dashed line in the ACIC 2022 coverage panel marks nominal 90% coverage.

### A.3.3. S3.3. DR PSEUDO-OUTCOME PEHE PROXY

For heterogeneous treatment effect estimation (IHDP, ACIC 2016, LaLonde-Norm), we construct cross-fitted doubly robust (DR) pseudo-outcomes following Kennedy (2024):

1. Split the data into $K = 2$ folds.
2. For each fold $k$, fit nuisance models on the complement $\{1, \ldots, K\} \setminus k$: outcome models $\hat{\mu}_t(x) = \mathbb{E}[Y \mid X = x, T = t]$ using both Ridge and histogram gradient-boosted regressors; propensity model $\hat{e}(x) = P(T = 1 \mid X = x)$ using logistic regression with clipping at $[0.05, 0.95]$; and a main-effect regression $\hat{m}(x) = \mathbb{E}[Y \mid X = x]$ using Ridge for the R-loss term.
3. Construct the DR pseudo-outcome for each unit $i$ in fold $k$: $\tilde{\tau}_i = \hat{\mu}_1(x_i) - \hat{\mu}_0(x_i) + T_i(Y_i - \hat{\mu}_1(x_i))/\hat{e}(x_i) - (1 - T_i)(Y_i - \hat{\mu}_0(x_i))/(1 - \hat{e}(x_i))$.
4. Proxy ITE fit:
$$\widehat{\mathrm{RMSE}}_{\tau,\text{proxy}} = \left( N^{-1} \sum_i (\hat{\tau}_i - \tilde{\tau}_i)^2 \right)^{1/2}.$$
Proxy ATE: $\hat{\tau}_{\text{AIPW}} = N^{-1} \sum_i \tilde{\tau}_i$.
5. We also compute a normalized R-loss term,
$$R_{\text{loss,norm}} = \frac{N^{-1} \sum_i \left(Y_i - \hat{m}(x_i) - (T_i - \hat{e}(x_i))\hat{\tau}_i\right)^2}{\widehat{\mathrm{Var}}(Y) + 10^{-8}}.$$
The implemented ITE proxy objective is
$$\mathcal{L}_{\text{ITE-proxy}} = 0.5\, \phi(\overline{\widehat{\mathrm{RMSE}}_{\tau,\text{proxy}}}) + 0.3\, \lambda_{\text{ATE}}\, \phi(\overline{|\hat{\tau} - \hat{\tau}_{\text{AIPW}}|}) + 0.2\, \phi(\overline{R_{\text{loss,norm}}}),$$
with $S_{\text{ITE-proxy}} = 1/(1 + \mathcal{L}_{\text{ITE-proxy}})$ and $\phi(x) = \log(1 + x)/(1 + \log(1 + x))$. As in Section S1.4, the overline denotes averaging across the training replicates used during search.

### A.3.4. S3.4. DR-DID COVERAGE PROXY (ACIC 2022)

For ACIC 2022's panel data setting, we developed a DR difference-in-differences pseudo-target. We fit outcome models for control units in the pre and post periods, fit a propensity model for treatment assignment (clipped to $[0.05, 0.95]$), and construct the DR-DID estimate per replicate. The most reliable proxy signal was whether a candidate estimator's confidence interval covered this DR-DID target. To prevent trivial high-coverage solutions based on overly wide intervals, we also tracked raw interval width and, when enough replicates were available, a secondary width-calibration term

$$g_{\text{width}} = \frac{\left| \overline{W} - 2 z_{0.95}\, \widehat{\mathrm{sd}}(\hat{\tau}_{\text{DR-DID}}) \right|}{2 z_{0.95}\, \widehat{\mathrm{sd}}(\hat{\tau}_{\text{DR-DID}})},$$

which compares the mean reported interval width with the nominal 90% width implied by the cross-replicate spread of the DR-DID targets. The implemented objective is

$$\mathcal{L}_{\text{ACIC-proxy}} = 0.6\, \phi(\overline{|\hat{\tau} - \hat{\tau}_{\text{DR-DID}}|}) + 0.25\, \lambda_{\text{cov}} \widetilde{C} + 0.15\, \phi(\overline{W}),$$

where $\widetilde{C}$ is the hit-rate gap alone when too few replicates are available for calibration, and otherwise a 70:30 blend of hit-rate centering and cross-replicate width calibration. Specifically, if $\widehat{p}_{\text{hit}}$ is the fraction of evaluated training replicates whose candidate interval covers the DR-DID target, then

$$\widetilde{C} = \begin{cases} \phi(|\widehat{p}_{\text{hit}} - 0.9|/0.9), & R < 2, \\ 0.7\, \phi(|\widehat{p}_{\text{hit}} - 0.9|/0.9) + 0.3\, \phi(g_{\text{width}}), & R \geq 2, \end{cases}$$

where $R$ is the number of evaluated training replicates. Across 76 checkpoint reevaluations, the DR-DID hit-rate component achieved $\rho = 0.99$ correlation with true coverage. We therefore treat DR-DID hit-rate as the strongest single ACIC proxy signal. Width calibration and raw interval width were nevertheless retained in the implemented scalarized objective as secondary regularizers against degenerate interval behavior, rather than as the main validated proxy component.

#### A.3.5. S3.5. PROXY VALIDATION SUMMARY

Supplementary Table 7 summarizes the alignment between the proxy objectives used during search and the true held-out metrics used for the main evaluation.

*Table 7.* **Proxy metric validation.** For IHDP, LaLonde-Norm, and ACIC 2016, values report Pearson correlation between matched final-run proxy combined scores and true combined scores. For ACIC 2022, values report Spearman correlation between the DR-DID hit-rate component and true coverage across checkpoint reevaluations.

| Dataset | Correlation | $n$ | $p$-value |
|---|---|---|---|
| IHDP | 0.97 | 4 | 0.03 |
| LaLonde-Norm | 0.77 | 4 | 0.23 |
| ACIC 2016* | 0.50 | 3 | 0.67 |
| ACIC 2022 (DR-DID hit-rate vs. true coverage) | 0.99 | 76 | $1.3 \times 10^{-67}$ |

*ACIC 2016 includes one additional cross-fitted ensemble proxy rerun (`_cf`) beyond the two standard matched proxy runs.

### A.4. S4. Dataset-level evolution trajectories

The four subsections below summarize representative best-run trajectories for each benchmark. Each begins by restating the identification problem faced by the agent, then lists the key transitions along the winning run, and finally describes the resulting evolved estimator in prose.

#### A.4.1. S4.1. ACIC 2022 TRAJECTORY

**Seed program and identification problem.** ACIC 2022 presents a panel data problem: the agent must estimate a patient-weighted SATT from practice-year data with pre/post periods. The seed program is a TWFE-DiD estimator with post-year interactions and cluster-robust WLS, generated by GPT-5 in the zero-shot sweep (RMSE $= 33.1$, coverage $= 0.40$).

**Key transitions.** Supplementary Table 8 details the feature progression across all new-best checkpoints.

*Table 8.* **ACIC 2022 feature progression.** Each row is a new-best checkpoint. The agent progressively discovers DiD logic, pre-mean adjustment, and trend correction.

| Iter. | RMSE | Cov. | Ridge | Cov. enc. | Pre-mean | Trend | Post-only | CI adj. |
|---|---|---|---|---|---|---|---|---|
| 0 (init) | 33.1 | 0.40 | — | — | — | — | — | — |
| 2 | 31.7 | 0.55 | ✓ | ✓ | — | — | — | — |
| 14 | 28.7 | 0.55 | — | ✓ | ✓ | — | — | — |
| 16 | 27.1 | 0.75 | — | ✓ | ✓ | — | — | ✓ |
| 22 | 18.5 | 0.95 | — | ✓ | ✓ | — | ✓ | ✓ |
| 52 | 15.3 | 0.90 | — | ✓ | ✓ | ✓ | ✓ | ✓ |
| 88 | 15.2 | 0.90 | — | ✓ | ✓ | ✓ | ✓ | ✓ |

**Final evolved program.** The best ACIC 2022 program ("DR DiD-RA SATT, cluster-robust") computes per-practice pre-period weighted means and slopes, differences $Y_{\text{post}} - \bar{Y}_{\text{pre}}$, selects the top 8 covariates by correlation with control-group $\Delta Y$, runs per-year ridge-regularized WLS regression adjustment on controls, computes patient-weighted ATT on residuals, and estimates cluster-robust variance using influence-function contributions with conservative (max of residual vs. raw) variance selection and 5% SE inflation.

#### A.4.2. S4.2. ACIC 2016 TRAJECTORY

**Seed program and identification problem.** ACIC 2016 is a cross-sectional benchmark with 58 covariates and mixed continuous/categorical features. The seed program is an X-learner with cross-fitted gradient-boosted outcome models and constant propensity weighting.

**Key transitions.** Supplementary Table 9 follows a representative best-run trajectory, in which the agent refines the seed X-learner by replacing constant propensity weighting with a random-forest propensity model and local propensity-weighted blending.

*Table 9.* **ACIC 2016 evolution checkpoints.** Best-run trajectory (Gemini, $\lambda = 0.2$, split 20:80). The agent refines a simple X-learner into a random-forest propensity-weighted X-learner with local blending.

| Iter. | Method | $\sqrt{\text{PEHE}}$ | \|ATE error\| | Combined |
|---|---|---|---|---|
| Init | X-Learner (GBM, cross-fitted) | 1.225 | — | baseline |
| Early | X-Learner + RF propensity (local weighting) | 1.028 | 0.105 | 0.579 |
| Final | X-Learner (GBM + RF PS) | 1.028 | 0.105 | 0.579 |

**Final evolved program.** The best ACIC 2016 program ("X-Learner, GBM + RF PS") replaces the seed's constant propensity weighting with a random forest propensity model, enabling local propensity-weighted blending $\hat{\tau}(x) = \hat{e}(x)\hat{\tau}_0(x) + (1 - \hat{e}(x))\hat{\tau}_1(x)$. It uses `pd.get_dummies` to handle the 58 mixed-type covariates, 5-fold cross-fitting for outcome surfaces, and separate second-stage GBM models for treated and control pseudo-outcomes. The learning rate is tuned to 0.07 (vs. 0.05 in the seed). A separate high-performing GPT-5 variant, not the trajectory shown in Supplementary Table 9, evolved further into a "Meta-Ensemble (T+DR+DRF, adaptive)" that combines three base learners—T-learner, DR pseudo-outcome, and DR-smoothed random forest—using inverse-variance meta-weights from cross-validated performance.

### A.4.3. S4.3. IHDP TRAJECTORY

**Seed program and identification problem.** IHDP is a semi-synthetic dataset with 25 covariates and known response surfaces. The seed program is identical to ACIC 2016: an X-learner with cross-fitted GBM outcome models and constant propensity weighting.

**Key transitions.** Supplementary Table 10 tracks a representative best-run trajectory, in which the agent augments the seed X-learner with an ensemble backbone, neural treatment-effect models, and adaptive propensity weighting.

*Table 10.* **IHDP evolution checkpoints.** Best-run trajectory (GPT-5, $\lambda = 0.2$, split 20:80). The agent augments the X-learner with an ensemble backbone, neural network treatment effect models, and adaptive propensity weighting.

| Iter. | Method | $\sqrt{\text{PEHE}}$ | \|ATE error\| |
|---|---|---|---|
| Init | X-Learner (GBM, cross-fitted, constant $g$) | 1.613 | 0.116 |
| Early | GBM+Ridge ensemble $(0.7/0.3)$ + polynomial features | $\approx$1.4 | $\approx$0.10 |
| Mid | + MLP treatment effect models (TARNet-style) | $\approx$1.3 | $\approx$0.10 |
| Final | TARNet-style X-learner + ensemble + adaptive-$g$ | 1.253 | 0.115 |

**Final evolved program.** The best IHDP program ("TARNet-style X-learner + ensemble + adaptive-$g$") makes three key changes from the seed: (1) *Ensemble outcome models*: cross-fitted 70/30 blends of GBM and Ridge regression replace pure GBM, improving stability on the small IHDP samples. (2) *Neural network treatment effect models*: the second-stage $\hat{\tau}_1, \hat{\tau}_0$ models use MLPRegressor (hidden layers 50–25, L2 regularization $\alpha = 0.01$) with GBM fallback, capturing nonlinear heterogeneity. (3) *Feature-dependent propensity weighting*: rather than the constant $g = \bar{T}$ used in the seed, the program first computes a per-unit feature score $f_i = p^{-1} \sum_{j=1}^{p} |X_{ij}^{\text{scaled}}|$, the mean absolute value of the standardized covariates for unit $i$, then min-max normalizes $f_i$ across units and sets $g_i = \text{clip}(g + 0.1(f_i - 0.5), 0.05, 0.95)$ for propensity-weighted blending of the X-learner arms. Additionally, the program adds interaction features between the top 5 scaled covariates and a propensity proxy, expanding the feature space from 25 to 30 dimensions.

### A.4.4. S4.4. LALONDE TRAJECTORY

**Seed program and identification problem.** LaLonde is an observational job-training study with only 8 covariates but severe selection bias between experimental participants and survey-based controls. The seed program is the same X-learner used for IHDP and ACIC 2016.

**Key transitions.** Supplementary Table 11 follows a representative best-run trajectory in which the agent blends X-learner and R-learner components with AIPW calibration.

**Final evolved program.** The best LaLonde program ("CF X+R blend, AIPW-calib, GBR+Ridge") is the most architecturally complex of the four datasets, reflecting the difficulty of observational bias correction. It combines:

*Table 11.* **LaLonde evolution checkpoints.** Best-run trajectory (GPT-5, $\lambda = 0.2$, split 20:80). The agent blends X-learner and R-learner components with AIPW calibration.

| Iter. | Method | $\sqrt{\text{PEHE}}$ | \|ATE error\| |
|---|---|---|---|
| Init | X-Learner (GBM, cross-fitted, constant $g$) | — | — |
| Early | + Cross-fitted propensity model + DR pseudo-outcomes | ≈1.5 | ≈0.20 |
| Mid | X-learner + R-learner blend with AIPW | ≈1.4 | ≈0.17 |
| Final | CF X+R blend, AIPW-calib (GBR+Ridge) | 1.359 | 0.161 |

- *Cross-fitted nuisance estimation*: 5-fold cross-fitting for outcome models (GBM, 160 trees, depth 3, subsample 0.8) and propensity scores (logistic regression with $C = 1.0$, clipped to $[0.02, 0.98]$).
- *X-learner component*: standard two-stage X-learner with GBM second-stage models and propensity-weighted blending $\hat{\tau}_X = \hat{e}\hat{\tau}_0 + (1 - \hat{e})\hat{\tau}_1$.
- *R-learner component*: residual-on-residual regression $Z_R = (Y - \hat{\mu}_{\text{mix}})/(T - \hat{e})$ with Ridge regression ($\alpha = 1.0$), weighted by $(T - \hat{e})^2$.
- *Ensemble blending*: 60% X-learner + 40% R-learner, calibrated so that $\bar{\hat{\tau}} = \hat{\tau}_{\text{AIPW}}$ via an additive shift.
- *Winsorization*: ITE predictions clipped at the 1st and 99th percentiles.

The evolution of an R-learner component is notable because the R-learner was not present in the seed program and represents a genuinely different identification strategy (residual-based) from the X-learner (imputation-based). The 60/40 blend weight and AIPW calibration were discovered through iterative mutation rather than being pre-specified.

### A.5. S5. Representative code edits during evolution

This final section reproduces short diffs from major inflection points in the evolutionary runs. The goal is to make the trajectory descriptions above concrete by showing the kinds of code mutations associated with meaningful performance jumps.

Listings 1, 2, 3, 4, and 5 show representative mutations for ACIC 2022, IHDP, ACIC 2016, and LaLonde, respectively.

#### A.5.1. S5.1. ACIC 2022: Baseline → Iteration 2 (ridge regularization)

*Listing 1.* ACIC 2022: Ridge regularization and covariate encoding added.

```
@@
-import statsmodels.api as sm
+import statsmodels.api as sm
+from sklearn.preprocessing import StandardScaler
+from sklearn.linear_model import Ridge

@@
-    req = ['id.practice', 'year', 'Y', 'Z', 'post', 'n.patients']
-    for c in req:
-        if c not in df.columns:
-            raise ValueError(f"Missing required column: {c}")
+    d = df.copy()
+    covariate_cols = [col for col in d.columns if col.startswith(('X','V'))]
+    for col in covariate_cols:
+        if d[col].dtype == 'object' or d[col].dtype.name == 'category':
+            dummies = pd.get_dummies(d[col], prefix=col, drop_first=True,
+                                     dtype=float)
+            d = pd.concat([d, dummies], axis=1)
```

#### A.5.2. S5.2. ACIC 2022: Iteration 16 → 22 (post-period restriction)

*Listing 2.* ACIC 2022: Pivotal mutation retains pre-period anchoring and restricts estimation to post-period differences.

```
@@
-    post_years = sorted(d.loc[d['post']==1, 'year'].unique().tolist())
+    pre = d.loc[d['post'] == 0].copy()
+    pre_means = (pre.groupby('id.practice')
+                     .apply(lambda g: np.average(g['Y'],
+                                                 weights=g['n.patients']))
+                     .rename('Y_pre_mean').reset_index())
+    d = d.merge(pre_means, on='id.practice', how='left')
+    post = d.loc[(d['post']==1) & np.isfinite(d['Y_pre_mean'])].copy()
+    post['D'] = post['Y'].astype(float) - post['Y_pre_mean'].astype(float)
```

### A.5.3. S5.3. IHDP: SEED → BEST (ENSEMBLE OUTCOME MODELS + NEURAL TREATMENT EFFECTS)

*Listing 3.* IHDP: Seed X-learner evolves to TARNet-style ensemble.

```
@@
-from sklearn.ensemble import GradientBoostingRegressor
+from sklearn.ensemble import GradientBoostingRegressor
+from sklearn.linear_model import Ridge
+from sklearn.neural_network import MLPRegressor

@@  # Cross-fitted outcome models
-        m0 = GradientBoostingRegressor(**params)
-        m0.fit(X_train[control_idx], y_train[control_idx])
-        mu0_preds[test_idx] = m0.predict(X_test)
+        gbr0 = GradientBoostingRegressor(**gbr_params)
+        ridge0 = Ridge(alpha=1.0, random_state=42)
+        gbr0.fit(Xtr[c_mask], ytr[c_mask])
+        ridge0.fit(Xtr[c_mask], ytr[c_mask])
+        mu0_oof[te] = 0.7 * gbr0.predict(Xte) + 0.3 * ridge0.predict(Xte)

@@  # Second-stage treatment effect models
-    tau0 = GradientBoostingRegressor(n_estimators=100, ...)
-    tau0.fit(X_scaled[control_idx], d0[control_idx])
+    tau1_nn = MLPRegressor(hidden_layer_sizes=(50, 25), alpha=0.01, ...)
+    tau1_nn.fit(X_enhanced[t == 1], d1[t == 1])

@@  # Propensity weighting
-    g = np.mean(treatment)
-    ite_hat = g * tau1_pred + (1 - g) * tau0_pred
+    feature_weight = np.mean(np.abs(X_scaled), axis=1)
+    feature_weight = (feature_weight - feature_weight.min()) / (...)
+    adaptive_g = g + 0.1 * (feature_weight - 0.5)
+    adaptive_g = np.clip(adaptive_g, 0.05, 0.95)
+    ite_hat = adaptive_g * tau1_pred + (1 - adaptive_g) * tau0_pred
```

### A.5.4. S5.4. ACIC 2016: SEED → BEST (LOCAL PROPENSITY WEIGHTING)

*Listing 4.* ACIC 2016: Constant propensity replaced by RF propensity model.

```
@@
-from sklearn.ensemble import GradientBoostingRegressor
+from sklearn.ensemble import GradientBoostingRegressor, RandomForestClassifier

@@  # Propensity score: global mean -> local RF estimate
-    g = np.mean(treatment)
-    g_pred = np.full(len(df), g)
-    ite_hat = g_pred * tau1_pred + (1 - g_pred) * tau0_pred
+    ps_model = RandomForestClassifier(n_estimators=100, max_depth=5,
+                                      min_samples_leaf=10, random_state=42)
+    ps_model.fit(X, t)
+    ps = np.clip(ps_model.predict_proba(X)[:, 1], 0.05, 0.95)
+    ite_hat = ps * tau0_x + (1 - ps) * tau1_x
```

### A.5.5. S5.5. LALONDE: SEED → BEST (R-LEARNER + AIPW CALIBRATION)

*Listing 5.* LaLonde: X-learner augmented with R-learner and AIPW calibration.

```
@@  # New: R-learner component (not in seed)
+    mu_mix = ehat * mu1 + (1 - ehat) * mu0
+    y_res = y - mu_mix
+    t_res = t - ehat
+    z_r = np.where(np.abs(t_res) > 1e-3, y_res / t_res, 0.0)
+    w_r = np.clip(t_res ** 2, 1e-4, None)
+    r_model = Ridge(alpha=1.0, random_state=42)
+    r_model.fit(Xs, z_r, sample_weight=w_r)
+    tau_r = r_model.predict(Xs)

@@  # New: Ensemble blend + AIPW calibration (not in seed)
+    ite_hat = 0.6 * tau_x + 0.4 * tau_r
+    shift = ate_dr - float(np.mean(ite_hat))
+    ite_hat = ite_hat + shift  # calibrate to AIPW ATE
```