

An Auditable AI Agent Loop for Empirical Economics

A Case Study in Forecast Combination

Minchul Shin

Federal Reserve Bank of Philadelphia

WP 26-41

PUBLISHED

August 2026

ISSN: 1962-5361

Disclaimer: This Philadelphia Fed working paper represents preliminary research that is being circulated for discussion purposes. The views expressed in these papers are solely those of the authors and do not necessarily reflect the views of the Federal Reserve Bank of Philadelphia or the Federal Reserve System. Any errors or omissions are the responsibility of the authors. Philadelphia Fed working papers are free to download at: <https://www.philadelphiafed.org/search-results/all-work?searchtype=working-papers>.

DOI: <https://doi.org/10.21799/frbp.wp.2026.41>

An Auditable AI Agent Loop for Empirical Economics: A Case Study in Forecast Combination

Minchul Shin
*Federal Reserve Bank of Philadelphia**

First version: March 17, 2026
This version: August 4, 2026

Abstract

AI coding agents, general-purpose assistants that write and execute code, make empirical specification search fast and cheap, but they also widen hidden researcher degrees of freedom. This paper adapts an open-source agent-loop architecture to an empirical economics workflow and adds a post-search holdout evaluation. In a forecast-combination illustration, independent agent searches find methods that improve on benchmarks from the original study. Logged search and holdout evaluation together make adaptive specification search more transparent and help distinguish robust improvements from sample-specific discoveries.

Keywords: Agent loops, Autoresearch, Specification search, Researcher degrees of freedom, Responsible AI, Forecast combination

JEL Codes: C53, C52, C18

*The views expressed herein are those of the author and do not necessarily reflect the views of the Federal Reserve Bank of Philadelphia or the Federal Reserve System. Email: visiblehand@gmail.com.

1 Introduction

Empirical researchers have always explored alternative specifications; AI coding agents, general-purpose assistants such as Claude Code, Codex, or OpenCode that write and execute code, make that exploration cheap and fast. The change is not only speed. Conventional specification search usually ranges over a menu fixed in advance, such as regressors or functional forms in $y = x'\beta + u$. Agentic search instead ranges over code. The researcher fixes the inputs, the object to be produced, and the score: for example data (y, x) , predictions $\hat{y}$, and out-of-sample MSE. The agent can then write code that maps the inputs into predictions, such as $\hat{y} = g(x; \hat{\beta})$. The searched object is therefore the code itself, not only a pre-listed formula or tuning parameter.

Researcher degrees of freedom widen accordingly, and the boundary between coding assistance and undisclosed specification search becomes thin. For a field long concerned with such degrees of freedom (Leamer, 1983; White, 2000; Gelman and Loken, 2013; Miguel, 2021), the central question is not whether AI agents can improve an empirical score, but whether their search can be made transparent and auditable.

Building on Karpathy’s (2026) open-source `autoresearch`, this paper adapts a simple agentic coding loop into a research protocol for empirical economics. The protocol has four ingredients: a written instruction contract, an immutable evaluator, a single editable script, and a complete experiment log. This paper also adds an explicit post-search holdout evaluation. Together, these elements make agentic search auditable: they record what was tried, limit where and how much the agent can search, and test the result beyond the search sample.

The empirical illustration applies this agentic-search protocol to the forecast-combination problem studied by Diebold and Shin (2019). The agent searches for improvements to the method proposed there. On the original rolling evaluation, three independent search runs beat existing methods. A post-search holdout for 2017Q1–2025Q4 then evaluates the discovered rules: two continue to outperform standard benchmarks, while one does not. The logs show

Algorithm 1 Auditable agent-loop protocol

Require: Instruction contract C , evaluator $S(\cdot; D^S)$, search sample D^S , initial script τ_0 , budget K

- 1: Run evaluator on τ_0 ; record score; set $\tau^{\text{best}} \leftarrow \tau_0$
 - 2: **for** $k = 1, \dots, K$ **do**
 - 3: Agent proposes candidate script τ_k , informed by $\log L_{k-1}$
 - 4: Run evaluator on τ_k ; record score (or crash)
 - 5: If score improves, set $\tau^{\text{best}} \leftarrow \tau_k$; otherwise revert to τ^{best}
 - 6: Append $(k, \tau_k, s_k, \text{outcome})$ to $\log L_k$
 - 7: **end for**
-

how each search moved from the initial task toward broader method discovery; the holdout shows which discoveries survive outside the search sample.

Recent AI-for-science work ranges from code rewriting against automated evaluators (Novikov et al., 2025; Nam et al., 2025) to broader research automation (Lu et al., 2024; Gottweis et al., 2025); in economics, Korinek (2025) surveys AI agents, and Dawid et al. (2025) develop agentic workflows. We study evaluator-locked local search in a fixed workflow.

2 An auditable agent loop, in practice

The protocol is easiest to see as a small modification of ordinary empirical work. Hold fixed the empirical design: the data, sample, information set, and scoring rule. Put the specification choice in one script. A researcher could now edit that script, rerun the fixed pipeline, compare the new score with the old one, and keep the change if it helps. The agent loop automates exactly this edit–evaluate–compare routine. Its output is not only the best script found, but the full trail of scripts tried and scores obtained. Algorithm 1 writes this routine in notation; the next subsection maps that notation to four working-directory files.

2.1 From empirical workflow to four files

The notation in Algorithm 1 is implemented through four working-directory objects. The instruction contract C is the researcher’s written design document and, operationally, the

Table 1: The four files of the protocol

File	Regression example	This paper’s application
Instruction contract C (program.md)	Define the search: improve validation fit in $y_i = x_i'\beta + g(z_i) + u_i$; edit only the code for g ; hold the sample and scoring rule fixed; stop after K attempts.	Find a tuning-parameter selection rule for peLASSO; edit only train.R ; soft budget of roughly 200 experiments. Verbatim copy archived in Appendix B.
Editable script τ (train.R)	The specification being searched: controls, transformations, interactions, shrinkage rules, or other code producing $\hat{g}(z_i)$. Each version is one candidate specification.	A function that maps the forecaster panel available at each forecast origin into one combined forecast using peLASSO with the candidate tuning-parameter selection rule.
Immutable evaluator S (prepare.R)	Holds fixed the empirical design. It runs each candidate script on the same data split and returns one score, such as validation MSE.	Calculates out-of-sample RMSE for the current train.R on the fixed search sample, passing only past data to train.R .
Experiment log L (results.tsv)	Records every specification tried, its score, and whether it was kept, discarded, or crashed.	A tab-separated values file with about 200 rows, as detailed in Section 3.

Notes: The file extensions are incidental. **.md** is a Markdown text file; any readable text format, such as **.txt**, could serve as the instruction contract. **.R** is an R program; it could instead be **.py**, **.m**, **.do**, or any runnable script in the researcher’s environment. **.tsv** is a tab-separated table; any editable table format, such as **.csv**, could serve as the log.

prompt read by the coding agent. It states the objective, context, rules, and experiment budget. The editable script τ contains the specification being searched. The immutable evaluator $S(\tau; D^S)$ holds fixed the empirical design, including the search sample D^S , and returns the score. The log L records every attempt. Table 1 maps these objects to files, first through a familiar regression-specification search and then through the forecast-combination application studied below.

This organization makes the adaptive search auditable by answering three questions. First, what was allowed to change? The file structure makes key restrictions machine checkable: the agent may edit **train.R**, but not the evaluator, and the evaluator controls which data are passed to the script. Any attempt to change another part of the codebase can be detected mechanically and treated as a failed experiment. Second, what was tried? Every candidate receives a mechanical status, *keep*, *discard*, or *crash*, and remains in the log even if it is later

reverted.¹ Third, how did the search arrive at the final script? Because the log is ordered and later proposals are informed by earlier successes and failures, the researcher can inspect the path from the baseline script to the selected one, including where the search improved, stalled, or drifted.

The separation between immutable search design and editable implementation parallels a pre-analysis plan, except that the audit trail is automatic and complete. The architecture applies whenever evaluation can be separated from implementation; Appendix F gives regression, structural VAR, and inflation-forecasting sketches, and Appendix G provides a more technical exposition of the search problem.

2.2 Running the loop, and the holdout rule

No custom software is required. The researcher opens a general-purpose coding assistant, such as Claude Code, Codex, or OpenCode, and asks it to implement the instruction contract. This step can be as simple as prompting “Read program.md and implement.” The agent then reads the contract, edits the allowed script, runs the fixed evaluator, logs the result, and repeats the loop in Algorithm 1 autonomously until the budget is exhausted.

After the search, the selected script can be evaluated on a post-search holdout D^H . The holdout rule is simple: D^H is scored once, by the researcher, after the search has ended. It is not used by the evaluator and is not available in the agent’s workspace. In our application, the holdout was assembled only after the agentic search was complete, so no holdout data existed in the working directory during the search.

2.3 Scope and caveats

Several caveats bound what the protocol can promise. First, residual look-ahead remains possible: a language model may encode background knowledge overlapping the holdout

¹In our application, each candidate is also preserved as a git commit, so logged rows can be recomputed after the fact rather than trusted.

period, so a holdout that is clean in the workspace need not be clean in the model’s memory. Holdout results are therefore a guardrail rather than proof. Second, the protocol disciplines search, not inference; it is not a substitute for formal post-selection inference. Third, the implementation is illustrative rather than optimal. The greedy keep-or-discard loop could be replaced by evolutionary or tree search (Novikov et al., 2025; Aygün et al., 2025).

Finally, the protocol does not guarantee semantic compliance. Written instructions do not bind an agent the way executable code does: the loop rewards scripts that improve the evaluator’s score, while many substantive restrictions in the contract are stated in natural language. For example, instructions such as “do not include higher order terms” or “do not combine with other methods” are semantic, and the agent may trade them off against score improvements in an opaque way. The search can therefore drift from the researcher’s intended model class. This is why auditability is essential: the protocol records the search path, so such drift is visible rather than hidden.

3 Empirical illustration: Real-time tuning of peLASSO

3.1 The tuning problem in Diebold and Shin (2019)

We consider a forecast-combination problem. There are K forecasts for the same target, and the goal is to construct a combined forecast, a function of these forecasts, with lower out-of-sample RMSE. In this setting, the simple average, which assigns equal weights in a linear aggregation, is famously hard to beat with estimated weights. One explanation is that estimation uncertainty can dominate the gains from estimating combination weights. This motivates regularized combination methods.

Diebold and Shin (2019)’s partially egalitarian LASSO (peLASSO) selects forecasters using LASSO, then shrinks the surviving weights toward equality. Implemented in real time, it requires choosing two regularization parameters at each forecast origin. The original paper instead reports ex post optimal parameters and proposes easier-to-tune subset averaging rules

inspired by the peLASSO analysis, leaving no data-based rule for real-time peLASSO tuning. On their euro area GDP data, ex post peLASSO attains RMSE 1.400 versus 1.504 for the simple average, a seven percent oracle gap. The present application takes this unresolved tuning problem as the object of agentic search.

3.2 Search design and samples

We keep this out-of-sample evaluation setting. The target is year-on-year euro area real GDP growth, and the inputs are $K = 23$ individual SPF forecasts over the 1999Q3–2016Q4 sample. The score is RMSE over 65 forecast dates, 2000Q4–2016Q4, matching the study’s burn-in convention.

Our starting point is the exact codebase that reproduces the study. We make two modifications to implement the protocol of Section 2:

1. *Score function* (*prepare.R*). This function calculates out-of-sample RMSE for a given estimation function, `train.R`. If `train.R` takes the simple average of the current forecasts, the score function exactly reproduces the original simple-average benchmark.
2. *Instruction contract* (*program.md*). We write the contract to instruct the agent: “Find the best look-ahead-free, data-based lambda selection algorithm for [peLASSO] forecast combination.”²

Together with `train.R` as the editable script and `results.tsv` as the experiment log, these files implement the four objects in Table 1. We run three independent searches, totaling 661 experiments. As benchmarks, we retain the simple average, ex post peLASSO, and the two strongest subset averaging methods in that paper: $\text{best} \leq 6\text{-average}$ and $\text{best} (\leq 6, \leq 40)\text{-average}$ (Tables 4 and 5).

²The full instruction contract is about 835 words because it also gives project context, describes the replicated study and directory structure, states the edit and no-look-ahead rules, and specifies logging and recovery instructions. Appendix B reproduces the exact verbatim contract.

Table 2: Search-sample and holdout evaluation

	Search sample		Holdout		Holdout excl. COVID	
	RMSE	Relative	RMSE	Relative	RMSE	Relative
Simple average	1.504	1.000	2.979	1.000	1.120	1.000
peLASSO ex post	1.400	0.930	2.964 [.191]	0.995	1.075 [.136]	0.960
Best ≤ 6 -avg	1.435	0.954	2.901 [.240]	0.974	1.043 [.347]	0.932
Best ($\leq 6, \leq 40$)-avg	1.378	0.916	2.901 [.127]	0.974	1.142 [.602]	1.020
Run 1 (232 experiments)	1.291	0.858	2.816 [.177]	0.945	0.965 [.234]	0.861
Run 2 (228 experiments)	0.767	0.510	2.417 [.127]	0.811	0.827 [.108]	0.739
Run 3 (201 experiments)	1.216	0.808	3.244 [.852]	1.089	1.305 [.755]	1.165

Notes: RMSE of forecast errors for euro area real GDP growth (year on year). Search sample: 1999Q3–2016Q4 data; RMSE scored over 65 forecast dates (2000Q4–2016Q4). Holdout: 2017Q1–2025Q4 (36 quarters). Holdout excl. COVID: holdout dropping 2020Q1–Q4 (32 quarters). Relative RMSE is computed against the simple average on each respective sample. Best ≤ 6 -avg and best ($\leq 6, \leq 40$)-avg are from the original paper, Tables 4 and 5. Bracketed values in the holdout columns are one sided p -values from the Diebold–Mariano test that the method has superior predictive accuracy relative to the simple average, using the EWC fixed- b approximation (Shin and Schor, 2026); these have intrinsically low power given the small holdout sample and serial correlation in forecast errors, and should be read as descriptive. peLASSO ex post re-optimizes λ on each evaluation window separately; see Appendix E for the fixed- λ variant. Full results including intermediate methods are in Appendix E.

After the search, we perform holdout evaluation. The split follows naturally from the replication boundary: that same codebase contains data only through 2016Q4 and uses the same out-of-sample rule. We extended the series to 2017Q1–2025Q4 (36 quarters) only after the agentic search ended, so these later realizations form a natural post-search holdout.

3.3 Empirical findings

Final scores. Table 2 reports RMSE for the search sample and the holdout. On the search sample, all three runs beat both the simple average and the ex post peLASSO benchmark, with relative RMSEs of 0.858, 0.510, and 0.808 for Runs 1–3.

On the holdout, Run 2 remains strongest, with relative RMSE 0.811 (0.739 excluding the COVID quarters of 2020), and Run 1 also holds up, at 0.945 (0.861 excluding COVID). The two benchmark methods from the original paper generalize modestly, each at 0.974. Thus, even on the holdout, Runs 1 and 2 outperform every non-agent benchmark in Table 2,

including the best methods from the original paper. In contrast, Run 3 (1.089) performs worse than the simple average on the holdout, and peLASSO ex post shows essentially no improvement (0.995). Agentic search can uncover methods that generalize, but also sample-specific improvements.

What the log reveals. The log shows how the agent moves from tuning to method discovery.³ Each run begins with the same tension: the contract says “Do not try to combine with other methods,” but also encourages “combining previous near-misses” and “more radical approaches” if progress stalls. Early attempts stay close to peLASSO tuning, trying cross-validation and related lambda-selection rules. As gains become harder, the paths diverge. Run 1 moves toward stability selection and performance weighting. Run 2 stops treating lambda choice as the only margin and builds a ranking-based combination with bias correction. Run 3 stays closest to peLASSO tuning, using adaptive LASSO with forward cross-validation, but eventually adds an egalitarian elastic-net blend. The useful information is not only that drift occurred, but when it occurred and in what direction. Final scores alone would hide this path; the log makes the tradeoff between semantic instructions and score improvement visible.

What we learn. Agentic search is powerful and useful for facilitating empirical research by automating specification search at scale. In our experiment, it found methods that beat standard benchmarks and, in two runs, survived a post-search holdout. It is also an adaptive search over many programs, so search-sample improvements can overfit and semantic instructions can be traded off against score improvements. The lesson is not to trust the final script alone. The useful object is the auditable bundle: fixed evaluator, confined edit surface, complete log, and a holdout scored once after the search.

³The appendices archive the full instruction contract, a detailed log excerpt, method details, and additional holdout and robustness checks.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work, the author used Claude Opus 4.6 and GPT 5.4 for editorial and programming assistance, including code cleanup and revision of parts of the manuscript draft; the agentic search experiments themselves used Claude Code as described in the paper. The author reviewed and edited the output as needed and takes full responsibility for the content of the published article.

References

- Aygün, Eser, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston, et al. An AI system to help scientists write expert-level empirical software. *arXiv preprint arXiv:2509.06503*, 2025.
- Dawid, Herbert, Philipp Harting, Hankui Wang, Zhongli Wang, and Jiachen Yi. Agentic workflows for economic research: Design and implementation. *arXiv preprint arXiv:2504.09736*, 2025. doi: 10.48550/arXiv.2504.09736.
- Diebold, Francis X. and Minchul Shin. Machine learning for regularized survey forecast combination: Partially-egalitarian LASSO and its derivatives. *International Journal of Forecasting*, 35(4):1679–1691, 2019. doi: 10.1016/j.ijforecast.2018.09.006.
- Gelman, Andrew and Eric Loken. The garden of forking paths: Why multiple comparisons can be a problem, even when there is no “fishing expedition” or “p-hacking” and the research hypothesis was posited ahead of time. Unpublished manuscript, 2013.
- Gottweis, Juraj, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, et al. Towards an AI co-scientist. *arXiv preprint arXiv:2502.18864*, 2025. doi: 10.48550/arXiv.2502.18864.

- Karpathy, Andrej. autoresearch. GitHub repository, 2026. <https://github.com/karpathy/autoresearch>.
- Korinek, Anton. AI agents for economic research. NBER Working Paper 34202, National Bureau of Economic Research, 2025.
- Leamer, Edward E. Let’s take the con out of econometrics. *American Economic Review*, 73(1):31–43, 1983.
- Lu, Chris, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. doi: 10.48550/arXiv.2408.06292.
- Miguel, Edward. Evidence on research transparency in economics. *Journal of Economic Perspectives*, 35(3):193–214, 2021. doi: 10.1257/jep.35.3.193.
- Nam, Jaehyun, Jinsung Yoon, Jiefeng Chen, Raj Sinha, Jinwoo Shin, and Tomas Pfister. DS-STAR: Data science agent for solving diverse tasks across heterogeneous formats and open-ended queries. *arXiv preprint arXiv:2509.21825*, 2025. doi: 10.48550/arXiv.2509.21825.
- Novikov, Alexander, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. doi: 10.48550/arXiv.2506.13131.
- Shin, Minchul and Nathan Schor. ForeComp: An R package for comparing predictive accuracy using fixed-smoothing asymptotics. *arXiv preprint arXiv:2603.07458*, 2026. doi: 10.48550/arXiv.2603.07458.
- White, Halbert. A reality check for data snooping. *Econometrica*, 68(5):1097–1126, 2000. doi: 10.1111/1468-0262.00152.

Online Appendix

An Auditable AI Agent Loop for Empirical Economics

Not for publication

A Implementation details

A.1 Four-file architecture

The protocol requires only four files in a shared directory:

$$C \equiv \text{program.md}, \quad \tau \equiv \text{train.R}, \quad S \equiv \text{prepare.R}, \quad L \equiv \text{results.tsv}.$$

Section 2.1 of the main text describes the role of each file; this appendix records the implementation details. `prepare.R` loads the search sample, sources `train.R`, and returns the scalar score; the holdout data is entirely outside the agent’s workspace. `results.tsv` is a tab-separated audit log in which each row records a commit identifier, the score, the outcome status, and a verbal description of the strategy attempted. In the working directory, the git commit history preserves the actual code of every retained τ_k .⁴

The four-file architecture is adapted from Karpathy’s (2026) `autoresearch` repository.⁵ The contribution here is not the loop itself but its translation into a protocol for empirical economics: the immutable evaluator fixes what counts as success, the editable script defines the search surface, the instruction contract specifies the researcher’s permissions, and the audit log turns every attempted specification into a reportable object.

⁴The replication archive includes the TSV logs and final code but not the full git object store. Full recoverability from commit history requires access to the original working directory.

⁵See <https://github.com/karpathy/autoresearch>. The architecture is representative of a broader class of agent loops in which a language model acts as the optimizer within a fixed evaluation harness.

A.2 Triggering the agent loop

In the simplest case, the researcher opens a general-purpose coding assistant and prompts it to implement `program.md`. The agent reads the instruction contract, which describes the experiment loop of Algorithm 1 in plain language, and begins the edit–evaluate–log–retain/revert cycle autonomously.

In our application, we used an outer shell script (`run.sh`) to restart agent sessions when the model reached its context window limit. Each restart reads `results.tsv` and the git history to recover state, and the session prompt injects the remaining experiment budget: “You have a budget of $K_{\text{remaining}}$ more experiments. Stop after that.” This is how the budget K is communicated and enforced in practice. The outer script also provides an approximate stop by counting rows in `results.tsv` and refusing to launch a new session once the count reaches K ; a session already in progress may overshoot. The full `run.sh` is available in the replication archive.

A.3 Isolating the holdout

The post-search holdout D^H must remain entirely outside the agent’s workspace during the search. In our application, the holdout period (2017Q1–2025Q4) was constructed after the agentic search was complete, so no holdout data existed in the working directory while the agent was running. More generally, researchers should ensure that holdout data is not accessible to the agent during the search. Because current coding agents can access the internet and execute arbitrary code, an agent could in principle download or construct holdout period data on its own. Disabling internet access during the search, restricting the agent’s file system permissions, or simply withholding the holdout data from the working directory are practical safeguards. Without such boundaries, the transparency of the holdout evaluation cannot be guaranteed.

Table 3: Experiment counts

	Experiments	Keeps	Discards	Crashes
Run 1	232	42	188	2
Run 2	228	61	164	3
Run 3	201	46	152	3

Notes: “Keep” means the modification improved on the current best RMSE; “discard” means it did not; “crash” means the candidate script exceeded the 30-minute timeout or produced an error. The budget is enforced as a soft constraint; realized counts vary.

A.4 Actual run

Each run uses Claude Code with Opus 4.6 at default thinking effort, starting from a fresh agent with the same initial script (simple average) and an approximate budget of $K = 200$ experiments. The evaluator calls the forecasting function 66 times ($t = 5, \dots, 70$) but scores RMSE over 65 periods ($t = 6, \dots, 70$), matching the burn-in and evaluation design in Diebold and Shin (2019). Table 3 reports the experiment counts for the three agent runs. The budget is enforced as a soft constraint; the realized counts are 232, 228, and 201 for Runs 1, 2, and 3, respectively. If a candidate script exceeds 30 minutes of runtime, the experiment is terminated and logged as a crash.

B Instruction contract

The following is the verbatim `program.md` used in the empirical application of Section 3.

```
# autoresearch | PE Lasso lambda selection

## Goal

Find the best look-ahead-free, data-based lambda selection algorithm
for PE Lasso forecast combination.

PE Lasso forecast combination we mean partially egalitary lasso method
in the paper /original_paper/DieboldShinEgalitarianLasso.md
```

(both one-step conceptualization and two-step implementation).
For both cases, we have two regularization parameters.

IMPORTANT: Focus on two-step implementation of PE Lasso. Do not try to combine with other methods. I want to focus on finding a way to choose these regularization parameters for PE Lasso so that it becomes tuning free method. We could try bunch of different CV methods. Or, other data adaptive method automatically selects these tuning parameters. We understand it is difficult problem finding two regularization parameters from two different stages. But, this is exactly why we want to run this experiments.

The simple average benchmark has RMSE ~ 1.50 . The ex-post oracle (which cheats by using future data) achieves ~ 1.40 . Your job is to close that gap.

You decide what strategies to explore. The reference implementation is in 'original_program/R/' | read it for ideas.

Setup

1. Agree on a run tag with the user: propose a tag based on today's date (e.g. 'mar11'). The branch 'autoresearch/<tag>' must not already exist.
2. Create the branch: 'git checkout -b autoresearch/<tag>'
3. Read the in-scope files:
 - 'program.md' | this file (your instructions)
 - 'prepare.R' | fixed evaluation harness. Do not modify.
 - 'train.R' | the file you modify. Defines 'select_lambda(info)'.
4. Check session history: Read 'results.tsv' and 'git log' to understand what has been tried before. This is your memory across sessions.
5. Initialize results.tsv if it doesn't exist: create it with just the header row.
6. Confirm and go.

The rules

- One file: You only edit 'train.R'. It must define 'select_lambda(info)' returning 'list(forecast, method)'.
- No lookahead: At time t , only data from $1:(t-1)$ is available.

The 'info' object enforces this.

- No new packages: Only glmnet, forecast, and base R.
- Runtime: Each full run (66 evaluation periods) should complete in under 30 minutes.
- Deterministic: Use a fixed seed if randomness is needed.

What's in 'info'

```
info$X_train      # rolling window training forecasts [max(1,t-w):(t-1), 23]
info$y_train      # corresponding actuals
info$X_history    # full history [1:(t-1), 23]
info$y_history    # full history actuals
info$x_new        # current period forecasts [23]
info$lambda_grid  # 200-point lambda grid, exp(15) to exp(-15)
info$lambda_grid2 # same grid (for two-step methods)
info$t            # current time index
info$w            # window size (20)
```

All functions from 'prepare.R' are in scope: 'pelasso_lasso_avg()', 'pelasso_lasso_eridge()', 'pelasso_lasso_lasso()', 'fit_lasso()', 'fit_ridge()', 'fit_lasso()', 'fit_eridge()', 'cv_loo_bandwidth()', 'cv_loo_bandwidth_2step()', etc.

Benchmarks

Method	RMSE	Notes
Simple average	~1.50	Baseline (no lambda)
Ex-post optimal peLASSO(LASSO, Avg)	~1.40	Oracle uses future data
Good data-based method	1.42-1.48	Realistic target range

Output format

The script prints:

```
---
method:      <name>
rmse:        <value>
benchmark_rmse: <simple average RMSE>
relative_rmse: <ratio>
```

Extract the key metric: `'grep "^rmse:" run.log'`

Logging results

Log every experiment to `'results.tsv'` (tab-separated).

Do NOT commit this file.

commit	rmse	status	description
a1b2c3d	1.5012	keep	baseline simple average
b2c3d4e	1.4723	keep	L00 CV for pelasso_lasso_avg
c3d4e5f	1.4891	discard	BIC-based lambda selection
d4e5f6g	0.0000	crash	2D CV OOM on full grid

Session recovery

You may be starting a fresh session after a previous one was interrupted (context limit, crash, etc.). On every session start:

1. Read `'results.tsv'` and `'git log --oneline'` to see what has been tried.
2. Reconcile state: Compare the last `'results.tsv'` entry to the latest git commit.
 - If the latest commit is NOT in `'results.tsv'`, it was never evaluated. Run it now and log the result before moving on.
 - If `'train.R'` has uncommitted changes, they are leftover from an interrupted edit. Review and either commit or discard.
3. Identify the current best RMSE from `'results.tsv'` (the lowest `'rmse'` with status `'keep'`).
4. Continue experimenting from there.

The experiment loop

LOOP (until experiment budget is exhausted or interrupted):

1. Read the current state: `'train.R'`, `'results.tsv'`, `git log`
2. Edit `'train.R'` with a new lambda selection strategy
3. `'git commit -m "descriptive message"'`
4. Run: `'Rscript prepare.R > run.log 2>&1'`
5. Extract: `'grep "^rmse:" run.log'`
6. If empty (crash): `'tail -n 50 run.log'` to diagnose
7. Log to `'results.tsv'`

8. If RMSE improved -> keep the commit
9. If RMSE worse -> 'git reset --hard HEAD~1'
10. Repeat

Logging is step 7, BEFORE the keep/revert decision in steps 8-9. This ensures every experiment is recorded even if the session is interrupted right after.

Timeout: If a run exceeds 30 minutes, kill it and treat as crash.

Crashes: Fix trivial bugs (typos, off-by-one). If the idea is fundamentally broken, log crash and move on.

NEVER STOP: Do not pause to ask if you should continue. Run autonomously until your experiment budget runs out or you are manually interrupted. If you run out of ideas, re-read 'original_program/R/' for new angles, try combining previous near-misses, or try more radical approaches.

C Experiment log excerpt

Table 4 presents a selected excerpt from the 201-experiment log of Run 3. The trajectory illustrates a systematic exploration pattern. The agent begins with standard cross-validation for peLASSO (experiments 1–4), quickly moves to adaptive LASSO with ridge-based penalty weights and forward cross-validation (experiments 25–45), discovers that multi-horizon forward CV improves regularization parameter selection (experiment 86), introduces model averaging over near-optimal lambda values (experiment 113), and finally arrives at an ensemble that blends peLASSO with an egalitarian elastic net component (experiments 138–201). Each phase builds on the structural lessons of the previous one.

D Discovered methods

Each agent run produced a final method and several intermediate variants. The holdout evaluation in Section E reports results for the final method and two representative intermediates

Table 4: Experiment log excerpt: Run 3

Exp.	Status	RMSE	Description
1	keep	1.5043	baseline simple average
2	keep	1.4692	pelasso(lasso,avg) LOO CV B=0
3	crash	—	pelasso(lasso,avg) LOO CV B=1 – zero variance error
4	discard	1.9556	pelasso(lasso,eridge) 2D LOO CV coarse 30pt grid
25	keep	1.4455	pe(adalasso,avg) fwdCV exp 0.95 ridge penalty weights
45	keep	1.3839	pe(adalasso,avg) rolling window ridge penalties decay=0.6
86	keep	1.3539	pe(adalasso,avg) h=2 step ahead forward CV LOO ridge
113	keep	1.3202	pe(adalasso,avg) h=2 model averaging top 2pct lambdas
138	keep	1.2533	pe(adalasso,blend) 80pct PE + 20pct eridge ensemble
152	keep	1.2253	pe(adalasso,blend) 70/30 PE + eEN cv.glmnet LOO
201	keep	1.2159	pe(adalasso,blend) model avg 1pct h1(0.05)+h2(0.95)

Notes: Selected rows from the 201-experiment log of Run 3. Descriptions are the agent’s verbatim one-line summaries, truncated for display; they are reproduced with the agent’s own lowercase spelling of `lasso`, `pelasso`, and `adalasso`, which match the function names in the codebase. Experiment 1 is the initial simple average baseline. RMSE for crash entries is undefined; the implementation records them as zero in `results.tsv`, but they are excluded from all comparisons (formally, $s_k = +\infty$). Full logs are in the replication archive.

per run, selected from distinct algorithmic phases. We describe each method here.

As stated in Section 3.2, all methods operate within the same rolling window framework except where noted. At each forecast origin t , let $\mathbf{X}_t \in \mathbb{R}^{n_t \times K}$ and $\mathbf{y}_t \in \mathbb{R}^{n_t}$ denote the training window of forecaster predictions and GDP realizations, where $n_t = \min(t - 1, W)$, $W = 20$, and $K = 23$. Let $\mathbf{x}_t = (x_{1,t}, \dots, x_{K,t})'$ denote the new forecaster predictions for period t . Each method produces a combined forecast $\hat{y}_t$; most methods use only $(\mathbf{X}_t, \mathbf{y}_t, \mathbf{x}_t)$, with exceptions noted below.

D.1 Run 1: Stability selection with performance weighting

Run 1 (final). Stability selection with performance weighted aggregation.

1. Fit elastic net ($\alpha = 0.65$) with 5 fold CV on the training window. Define $\hat{\lambda}_{1.5se}$ as the largest λ such that the CV error is within 1.5 standard errors of the minimum.
2. *Stability selection.* Let $\mathcal{W} = \{j : |\log \lambda_j - \log \hat{\lambda}_{1.5se}| < \delta\}$ with $\delta = 0.25$. The selection

frequency of forecaster k is

$$\pi_k = |\mathcal{W}|^{-1} \sum_{j \in \mathcal{W}} \mathbf{1}\{\hat{\beta}_k(\lambda_j) \neq 0\}. \quad (1)$$

The active set is $\mathcal{A} = \{k : \pi_k > 0.4\}$.

3. *Composite weights.* Using the $m = 2$ most recent training observations, compute individual RMSE_k for each $k \in \mathcal{A}$ and form

$$w_k = \pi_k^2 \cdot \text{RMSE}_k^{-p}, \quad p = 14. \quad (2)$$

The exponent $p = 14$ was selected by the search from nearby alternatives (powers 2 through 20 were tried).

4. *Dominance check.* If $w_{(1)} > 5 \cdot w_{(2)}$ (where $w_{(j)}$ denotes the j th largest weight), use the single best forecaster.
5. Otherwise, the combined forecast is the weighted q th quantile ($q = 0.44$) of $\{x_{k,t}\}_{k \in \mathcal{A}}$ with normalized weights $w_k / \sum_k w_k$, computed by linear interpolation of the weighted empirical CDF.

Run 1a (Phase 2: Elastic net with median). PE selection using elastic net ($\alpha = 0.65$) with 5 fold CV and $\hat{\lambda}_{1\text{se}}$. The combined forecast is the median of the selected forecasters:

$$\hat{y}_t = \text{median}\left(\{x_{k,t}\}_{k \in \mathcal{A}(\hat{\lambda}_{1\text{se}})}\right). \quad (3)$$

No stability selection, no performance weighting.

Run 1b (Phase 1: peLASSO with CV). Standard peLASSO(LASSO, Avg) with $\hat{\lambda}_{1se}$ from `cv.glmnet`:

$$\hat{y}_t = |\mathcal{A}(\hat{\lambda}_{1se})|^{-1} \sum_{k \in \mathcal{A}(\hat{\lambda}_{1se})} x_{k,t}. \quad (4)$$

This is the closest variant to the original Diebold and Shin (2019) method, but with data driven λ selection rather than ex post optimization.

D.2 Run 2: Ranking, adaptive windows, and bias correction

Run 2 (final). Temporally weighted ranking with adaptive window selection and bias correction.

For a given training window $(\mathbf{X}, \mathbf{y})$ of size n :

1. *Temporal weights.* Define $\omega_s = \exp(-\alpha(n-s)/(n-1)) / \sum_{j=1}^n \exp(-\alpha(n-j)/(n-1))$ with $\alpha = 6$. More recent observations receive higher weight.
2. *Weighted MAE ranking.* $MAE_k = n^{-1} \sum_{s=1}^n n \omega_s |y_s - x_{k,s}|$. Rank forecasters by ascending MAE.
3. *LOO CV for subset size.* For each candidate $N \in \{3, \dots, \min(K, 18)\}$ and each left out observation i :
 - Rerank by RMSE on the remaining $n - 1$ observations.
 - Select the top N forecasters, weight them by $\tilde{w}_j \propto MAE_j^{-3}$ (on leave out data).
 - Compute the weighted average LOO forecast $\hat{y}_i^{LOO} = \sum_j \tilde{w}_j x_{j,i}$ and the median based LOO forecast $\hat{y}_i^{pct} = Q_{0.50}(\{x_{j,i}\}_{j \in \text{top } N})$.

Select $N^* = \arg \min_N \sqrt{n^{-1} \sum_i (\hat{y}_i^{LOO} - y_i)^2}$. After N^* is selected, the current period forecast (step 5) uses the full window weighted MAE ranking, not the LOO reranking or the MAE^{-3} weighted average.

4. *Adaptive window.* Evaluate steps 1–3 at the full rolling window and at each sub-window $w' \in \{4, 5, \dots, 19\}$, keeping whichever (N^*, w') combination yields the lowest LOO RMSE.
5. *Median aggregation.* The base forecast is $\tilde{y}_t = Q_{0.50}(\{x_{k,t}\}_{k \in \text{top } N^*})$.
6. *Bias correction.* The final forecast applies a bias adjustment based on the most recent leave one out error:

$$\hat{y}_t = \tilde{y}_t - \gamma \cdot \hat{e}_n^{\text{LOO,pct}}, \quad \gamma = 0.80, \quad (5)$$

where $\hat{e}_n^{\text{LOO,pct}} = \hat{y}_n^{\text{pct}} - y_n$ is the percentile based LOO error from the most recent training observation.

Run 2a (no bias correction). Identical to Run 2 (steps 1–5) but omits the bias correction (5):

$$\hat{y}_t = \tilde{y}_t. \quad (6)$$

This isolates the contribution of the forecaster ranking and adaptive window mechanism.

Run 2b (weighted ranking, no adaptive window). Steps 1–2 as in Run 2 with a fixed window. Step 3 uses a penalized LOO criterion, $\text{RMSE}_{\text{LOO}}(N) + 0.01 \cdot \log(N + 1)$, to select N^* . The combined forecast is a weighted average (not median) of the top N^* forecasters:

$$\hat{y}_t = \sum_{j=1}^{N^*} \tilde{w}_j x_{\sigma(j),t}, \quad \tilde{w}_j \propto \text{MAE}_{\sigma(j)}^{-3}, \quad (7)$$

where σ denotes the MAE ranking. No adaptive window, no bias correction.

D.3 Run 3: Adaptive LASSO with forward CV and ensemble

Run 3 (final). Adaptive LASSO with multi horizon forward CV, model averaging, and egalitarian elastic net ensemble.

1. *Pilot estimator.* Fit elastic net ($\alpha_{\text{pilot}} = 0.1$) with LOO CV on the rolling window to obtain $\hat{\beta}^{\text{ridge}}$.
2. *Adaptive LASSO penalties.* Define forecaster specific penalty factors

$$\tilde{w}_k = \frac{(|\hat{\beta}_k^{\text{ridge}}| + \epsilon)^{-1}}{(\bar{|\hat{\beta}^{\text{ridge}}|} + \epsilon)^{-1}}, \quad \epsilon = 5 \times 10^{-3}, \quad (8)$$

where $\bar{\cdot}$ denotes the cross-sectional mean. Fit LASSO with penalty $\lambda \cdot \tilde{w}_k |\beta_k|$ on the full history.

3. *Multi horizon forward CV.* Lambda is selected by walk forward CV on the full history preceding the current origin, rows $1, \dots, t-1$. For fold f , fit adaptive LASSO on the first t_f rows and compute forecast errors on the next two rows $t_f + 1$ and $t_f + 2$ (where $t_f + 2 \leq t - 1$ by construction). Let $e_{f,s}(\lambda) = \hat{y}_{t_f+s}(\lambda) - y_{t_f+s}$ for $s = 1, 2$, where $\hat{y}_{t_f+s}(\lambda)$ applies the λ dependent combination rule to the forecaster predictions at row $t_f + s$. Two candidate aggregation rules are evaluated at each λ : simple average and blended average ($0.7 \times \text{mean} + 0.3 \times \text{ridge weighted mean}$). The CV criterion is

$$\text{CV}(\lambda) = \sum_f w_f (0.05 \cdot e_{f,1}^2(\lambda) + 0.95 \cdot e_{f,2}^2(\lambda)), \quad (9)$$

with fold weights $w_f \propto \gamma^{F-f}$ and $\gamma = 0.75$.

4. *Model averaging.* Average forecasts over all λ within 1% of the best CV score, weighted by inverse RMSE.
5. *Egalitarian elastic net.* Fit elastic net ($\alpha = 0.5$) via `cv.glmnet` with LOO CV on the rolling window, selecting $\hat{\lambda}_{\min}$, on the egalitarian transformation $\tilde{y}_s = y_s - K^{-1} \sum_k x_{k,s}$, obtaining $\tilde{\beta}^{\text{eEN}}$, and compute $\hat{y}_t^{\text{eEN}} = \sum_k (\tilde{\beta}_k^{\text{eEN}} + K^{-1}) x_{k,t}$.
6. *Final blend.*

$$\hat{y}_t = 0.70 \cdot \hat{y}_t^{\text{PE}} + 0.30 \cdot \hat{y}_t^{\text{eEN}}. \quad (10)$$

Run 3a (Phase 4: Single horizon, no blend). Steps 1–2 as in Run 3. Forward CV evaluating only at the next panel row ($t_f + 1$) with $\gamma = 0.75$. No model averaging. Simple average of selected forecasters. No egalitarian elastic net blend:

$$\hat{y}_t = |\mathcal{A}(\hat{\lambda})|^{-1} \sum_{k \in \mathcal{A}(\hat{\lambda})} x_{k,t}. \quad (11)$$

Run 3b (Phase 5: Multi horizon, no blend). Same pilot and adaptive penalty construction as Run 3, with $\gamma = 0.80$ and $\epsilon = 10^{-3}$. Forward CV evaluates only the simple average aggregation rule (no blended alternative). Model averaging over λ values within 1% of the best CV score is applied. No egalitarian elastic net blend: $\hat{y}_t = \hat{y}_t^{\text{PE}}$.

E Post-search holdout evaluation

This section reports the full holdout evaluation summarized in Section 3 of the main text. The holdout period (2017Q1–2025Q4, 36 quarters) was withheld from the evaluator during the agentic search.

E.1 Data and processing

The extended dataset combines ECB Survey of Professional Forecasters (SPF) individual microdata with Eurostat GDP realizations (table `namq_10_gdp`, chain linked volumes, seasonally and calendar adjusted). The original sample (1999Q3–2016Q4, 70 observations) is copied verbatim from the original `H1_gdp.csv` of Diebold and Shin (2019) to guarantee exact RMSE reproduction. The new sample (2017Q1–2025Q4, 36 quarters) is constructed from the SPF extraction. Missing forecaster responses in the new sample are imputed by linear interpolation for interior gaps and by cross-sectional mean for trailing gaps and inactive forecasters. COVID quarters are defined as 2020Q1–2020Q4.

E.2 Full holdout results

Table 5 reports RMSE for all methods, including intermediate variants that isolate individual components. For each run, we include two representative intermediate methods selected from distinct algorithmic phases. The intermediate methods are informative because they isolate the contribution of individual components such as stability selection, bias correction, or ensemble blending.

Table 5: Full holdout evaluation of forecast combination methods

	Search sample		Holdout		Holdout excl. COVID	
	RMSE	Relative	RMSE	Relative	RMSE	Relative
<i>Panel A: Benchmarks and original paper methods</i>						
Simple average	1.504	1.000	2.979	1.000	1.120	1.000
peLASSO ex post (original)	1.400	0.930	3.172 [.880]	1.065	1.360 [.885]	1.214
peLASSO ex post (per window)	1.400	0.930	2.964 [.191]	0.995	1.075 [.136]	0.960
Best ≤ 6 -avg	1.435	0.954	2.901 [.240]	0.974	1.043 [.347]	0.932
Best ($\leq 6, \leq 40$)-avg	1.378	0.916	2.901 [.127]	0.974	1.142 [.602]	1.020
Best individual	1.403	0.933	2.994 [.716]	1.005	1.164 [.714]	1.040
<i>Panel B: Run 1 family</i>						
Run 1 (final)	1.291	0.858	2.816 [.177]	0.945	0.965 [.234]	0.861
Run 1a (ENet, median)	1.390	0.924	2.991 [.742]	1.004	1.158 [.794]	1.034
Run 1b (peLASSO, CV)	1.451	0.964	3.059 [.938]	1.027	1.270 [.929]	1.134
<i>Panel C: Run 2 family</i>						
Run 2 (final)	0.767	0.510	2.417 [.127]	0.811	0.827 [.108]	0.739
Run 2a (no bias corr.)	1.308	0.869	2.949 [.087]	0.990	1.093 [.290]	0.976
Run 2b (weighted avg)	1.351	0.898	2.918 [.113]	0.980	1.120 [.499]	1.000
<i>Panel D: Run 3 family</i>						
Run 3 (final)	1.216	0.808	3.244 [.852]	1.089	1.305 [.755]	1.165
Run 3a ($h=1$, no blend)	1.309	0.870	2.994 [.648]	1.005	1.238 [.818]	1.106
Run 3b ($h=2$, no blend)	1.307	0.869	3.037 [.865]	1.020	1.276 [.855]	1.140

Notes: RMSE of forecast errors for euro area real GDP growth (year on year). Search sample: 1999Q3–2016Q4 data; RMSE scored over 65 forecast dates (2000Q4–2016Q4). Holdout: 2017Q1–2025Q4 (36 quarters). Holdout excl. COVID: holdout dropping 2020Q1–Q4 (32 quarters). Relative RMSE is computed against the simple average on each respective sample. Bracketed values in the holdout columns are one sided p -values from the Diebold–Mariano test that the method has superior predictive accuracy relative to the simple average, using the EWC fixed- b approximation (Shin and Schor, 2026); these have intrinsically low power given the small holdout sample (36 quarters, 32 excluding COVID) and serial correlation in forecast errors, and should be read as descriptive. peLASSO ex post (original) and best individual use parameters selected on the search sample and are evaluated without re-optimization on the holdout. peLASSO ex post (per window) re-optimizes λ on each evaluation window separately.

E.3 Role of individual components

The intermediate methods isolate what drives generalization.

Stability selection (Run 1 vs. 1a, 1b). The full Run 1 (holdout relative RMSE 0.945) substantially outperforms both alternatives: Run 1a (1.004), which uses elastic net selection without stability, and Run 1b (1.027), which uses standard peLASSO with CV. The stability selection mechanism and the composite performance weighting are jointly responsible for the holdout gain. Without them, elastic net PE selection is essentially no better than the simple average.

Bias correction (Run 2 vs. 2a, 2b). Removing the bias correction from Run 2 degrades performance from 0.811 to 0.990 (Run 2a) or 0.980 (Run 2b). The bias correction contributes most of Run 2’s advantage. However, Runs 2a and 2b are still competitive with the simple average, indicating that the temporally weighted ranking and adaptive window mechanism also add modest value. The bias correction adjusts for persistent short run bias in combined forecast errors, which appears to reflect a structural feature rather than overfitting to the search sample.

Egalitarian elastic net blend (Run 3 vs. 3a, 3b). Run 3’s 70/30 blend with the egalitarian elastic net (holdout relative RMSE 1.089) performs worse than the simpler PE adaptive LASSO without the blend (3a at 1.005, 3b at 1.020). The ensemble component that improved search-sample fit introduced additional estimation noise that does not pay off on the holdout.

E.4 Independent dataset robustness

To verify that these results are not an artifact of using the original `H1_gdp.csv` verbatim for the search-sample period, we also constructed a fully independent dataset where all 106 rows are built from downloaded SPF and Eurostat data, with no verbatim copy from the original

file. The independent dataset uses the current (2026) GDP vintage rather than the 2018Q1 vintage in the original study; over the original sample the mean absolute revision is 0.079 pp and the maximum is 0.20 pp.

Table 6 repeats the exercise on this independent dataset. The key rankings are preserved: Run 2 achieves a relative RMSE of 0.808 on the holdout, Run 1 achieves 0.946, and the original paper methods achieve 0.974 and 0.968. Run 3’s relative RMSE shifts from 1.089 to 0.962 on the independent dataset, but its p -values remain very large in both cases, so the difference is not meaningful. The independent dataset confirms that the main ranking is preserved across vintages: Run 2 strongest, Run 1 second, and Run 3 not distinguishable from the simple average.

F Illustration: Beyond forecasting combination

As discussed in Section 2.1, the same four-file architecture applies whenever the researcher can separate an immutable evaluation design from an editable implementation rule. The remainder of this section provides three stylized illustrations that map the protocol into familiar empirical settings: regression specification search, structural VARs, and inflation forecasting for monetary policy.

These illustrations are intentionally schematic. In forecasting problems, the evaluator can often be a straightforward out-of-sample loss. In inference-heavy settings, by contrast, any scalar score should be understood as a diagnostic screen within a broader research design, not as a complete validation of identification or inference.

Example 1: Specification search in a regression model. Consider a researcher studying a descriptive wage regression using cross-sectional data. The estimating equation is

$$\log w_i = \beta \cdot \text{educ}_i + g(z_i) + e_i,$$

Table 6: Full holdout evaluation: Independent dataset

	Search sample		Holdout		Holdout excl. COVID	
	RMSE	Relative	RMSE	Relative	RMSE	Relative
<i>Panel A: Benchmarks and original paper methods</i>						
Simple average	1.510	1.000	2.979	1.000	1.120	1.000
peLASSO ex post (original)	1.417	0.938	3.041 [.763]	1.021	1.350 [.881]	1.206
peLASSO ex post (per window)	1.417	0.938	2.976 [.249]	0.999	1.119 [.188]	0.999
Best ≤ 6 -avg	1.500	0.994	2.901 [.239]	0.974	1.043 [.346]	0.931
Best ($\leq 6, \leq 40$)-avg	1.298	0.860	2.884 [.100]	0.968	1.094 [.340]	0.977
Best individual	1.407	0.932	2.909 [.245]	0.977	1.109 [.463]	0.990
<i>Panel B: Run 1 family</i>						
Run 1 (final)	1.295	0.858	2.817 [.177]	0.946	0.965 [.233]	0.861
Run 1a (ENet, median)	1.479	0.980	2.991 [.730]	1.004	1.157 [.779]	1.033
Run 1b (peLASSO, CV)	1.455	0.964	3.051 [.952]	1.024	1.250 [.931]	1.116
<i>Panel C: Run 2 family</i>						
Run 2 (final)	0.834	0.552	2.408 [.127]	0.808	0.826 [.107]	0.737
Run 2a (no bias corr.)	1.304	0.864	2.947 [.090]	0.989	1.096 [.298]	0.978
Run 2b (weighted avg)	1.343	0.890	2.898 [.100]	0.973	1.065 [.054]	0.950
<i>Panel D: Run 3 family</i>						
Run 3 (final)	1.305	0.864	2.865 [.309]	0.962	1.349 [.728]	1.205
Run 3a ($h=1$, no blend)	1.490	0.987	2.960 [.400]	0.994	1.270 [.802]	1.133
Run 3b ($h=2$, no blend)	1.502	0.995	2.963 [.414]	0.995	1.276 [.812]	1.139

Notes: Same as Table 5 but using the independently constructed dataset (all rows from downloaded SPF and Eurostat data, no verbatim copy from the original `H1_gdp.csv`). Search-sample RMSEs differ from Table 5 due to GDP vintage differences and independent imputation. Bracketed values in the holdout columns are one sided p -values from the Diebold–Mariano test using the EWC fixed- b approximation (Shin and Schor, 2026); see Table 5 notes for details and caveats. peLASSO ex post (original) and best individual use parameters selected on the search sample; see Table 5 notes for details.

where w_i is earnings, educ_i is years of schooling, β is a coefficient of interest, and z_i is a vector of controls including experience, experience squared, demographic indicators, occupation, industry, and geographic region. The researcher wants to compare alternative specifications for the nuisance function $g(\cdot)$ while handling inference on β separately.

File mapping.

- **program.md** (C): “Compare candidate specifications of $g(z_i)$. Only edit **train.R**. The search sample and its internal evaluation split are fixed in **prepare.R**. Do not modify the sample or the outcome variable.”
- **train.R** (τ): Defines a function that receives the training sample and returns fitted values for the nuisance component, potentially along with fitted values for educ_i if a partialling-out design is used. The agent may explore polynomial terms, splines, interactions, LASSO-penalized specifications, or other functional forms for g .
- **prepare.R** (S): Splits the search sample D^S into a training partition $\mathcal{I}_{\text{train}}$ and an evaluation partition $\mathcal{I}_{\text{eval}}$, fixed ex ante. On $\mathcal{I}_{\text{train}}$, it sources **train.R** to obtain the nuisance fit, and if desired partials out both outcome and treatment before estimating $\hat{\beta}$. On $\mathcal{I}_{\text{eval}}$, it computes a holdout prediction error such as

$$S(\tau; D^S) = \frac{1}{|\mathcal{I}_{\text{eval}}|} \sum_{i \in \mathcal{I}_{\text{eval}}} (\log w_i - \hat{\beta} \cdot \text{educ}_i - \hat{g}(z_i))^2.$$

- **results.tsv** (L_K): Records each specification attempted, its evaluation MSE, and the implied $\hat{\beta}$.

If the scoring rule were the in-sample p -value for $\hat{\beta}$, the loop would amount to automated p -hacking. The discipline comes from fixing the evaluator and the sample split before the search begins and from disclosing the full search path. Causal claims about returns to schooling would still require separate identification assumptions. After the search, the researcher can

evaluate the winning specification on a post-search holdout D^H (e.g., a later survey year or a pre-reserved subsample withheld from the search) to check whether the nuisance fit generalizes.

Example 2: Structural VAR identification. Consider a three-variable VAR for macroeconomic analysis of monetary policy. Let $y_t = (\Delta\text{GDP}_t, \pi_t, i_t)'$, where ΔGDP_t is real GDP growth, π_t is inflation, and i_t is the federal funds rate. The reduced-form VAR is

$$y_t = \Phi_1 y_{t-1} + \cdots + \Phi_p y_{t-p} + u_t,$$

where u_t is the reduced-form innovation with $\text{Var}(u_t) = \Sigma_u$. The structural representation is $u_t = A_0^{-1} \varepsilon_t$, where ε_t is the vector of structural shocks and A_0 encodes the contemporaneous causal structure. The goal here is more modest: compare candidate ways of constructing a monetary policy shock series and record how each one lines up with an external narrative diagnostic.

File mapping.

- **program.md** (C): “Compare candidate constructions of a monetary policy shock series using a fixed external narrative diagnostic. Only edit **train.R**. Do not modify the VAR specification.”
- **train.R** (τ): Defines a function that receives the estimated reduced-form parameters $(\hat{\Phi}_1, \dots, \hat{\Phi}_p, \hat{\Sigma}_u)$, together with any auxiliary diagnostic series supplied by **prepare.R**, and returns a candidate monetary policy shock series $\hat{\varepsilon}_t^{mp}$ and the associated GDP impulse response at horizon h . The agent may explore recursive orderings, sign restrictions, or other mappings from reduced-form innovations into candidate shock measures.
- **prepare.R** (S): Estimates the reduced-form VAR on a fixed training sample, calls

`train.R` to obtain the candidate shock, and computes a diagnostic score on D^S :

$$S(\tau; D^S) = - \left| \text{corr}(\hat{\varepsilon}_t^{mp}, z_t^{RR}) \right|,$$

where z_t^{RR} is the Romer and Romer narrative monetary policy shock measure. The negative sign ensures that higher correlation corresponds to a lower (better) score. This score is best read as an external diagnostic rather than a standalone proof of identification.

- `results.tsv` (L_K): Records the candidate construction, the diagnostic score, and the implied peak response of GDP to a monetary policy shock.

The point is not that the agent “finds the largest effect” or settles identification, but that any search over candidate shock constructions is visible, bounded, and judged against a criterion fixed ex ante. After the search, the researcher can evaluate the chosen shock construction on a later sample period D^H (e.g., search on 1969–2000, holdout on 2001–2007) to check whether the narrative correlation and impulse response patterns hold outside the search sample.

Example 3: Inflation forecasting for monetary policy. Central banks routinely produce inflation forecasts to guide monetary policy decisions. The workhorse framework is the Phillips curve:

$$\pi_{t+h} = \alpha + \beta \cdot \text{slack}_t + \gamma \cdot \pi_t^e + \delta' z_t + e_{t+h},$$

where π_{t+h} is h -quarter-ahead CPI inflation, slack_t is a measure of economic slack, π_t^e is an inflation expectations proxy, and z_t is a vector of additional conditioning variables (commodity prices, exchange rates, financial conditions). The practical difficulty is that the specification involves many choices: which slack measure (unemployment gap, output gap, capacity utilization), which expectations proxy (survey expectations, breakeven inflation, adaptive

expectations), which additional predictors, which estimation method (OLS, ridge, random forest), and which estimation window.

File mapping.

- **program.md** (C): “Compare candidate real-time Phillips curve specifications for $h = 4$ quarter-ahead CPI inflation forecasting. No lookahead: at forecast origin t , only real-time vintage data through t is available. Only edit **train.R**. Budget: $K = 200$ experiments.”
- **train.R** (τ): Defines a function that receives the information set $\mathcal{F}_t = \{y_s, x_s\}_{s=1}^t$ and returns a point forecast $\hat{\pi}_{t+h|t}$. The agent may explore different slack measures, expectations proxies, transformations, variable selection methods, penalized estimation, or nonlinear specifications.
- **prepare.R** (S): Implements pseudo-out-of-sample evaluation. For each forecast origin $t \in \{t_0, t_0 + 1, \dots, T - h\}$, passes the real-time information set to **train.R** and collects the forecast. The score is

$$S(\tau; D^S) = \text{RMSE} = \left(\frac{1}{|\mathcal{E}|} \sum_{t \in \mathcal{E}} (\pi_{t+h} - \hat{\pi}_{t+h|t})^2 \right)^{1/2}.$$

- **results.tsv** (L_K): Records each Phillips curve specification, its RMSE, and a description of the method (e.g., “ridge Phillips curve, unemployment gap, SPF expectations, 40-quarter rolling window”).

This is a natural fit for the protocol because the evaluator is a genuine pseudo-out-of-sample forecast loss. For central bank staff who routinely compare Phillips curve specifications, the protocol would make the search path more visible and easier to disclose. The researcher can further extend the evaluation to a later period D^H not available during the search to check whether the selected specification generalizes.

These three illustrations suggest how the four-file protocol can be adapted across settings. The architecture applies whenever the researcher can separate an immutable evaluation design from an editable implementation rule. Forecasting is the cleanest instance because the score is naturally out-of-sample. In more inference-heavy settings, the protocol is better viewed as a way to make adaptive search more visible and easier to disclose, not as a substitute for the underlying econometric argument.

A post-search holdout can also help reduce concerns about p-hacking, but only in a limited sense. Its value is that it separates discovery from evaluation: the agent may search adaptively over many specifications on the search sample, but the selected output is then judged on data not used during that search. This makes sample-specific improvements less likely to be mistaken for robust findings. At the same time, a holdout is not a complete solution. If the holdout is consulted repeatedly and the method is revised in response, it becomes part of the search process itself. And in inference-heavy settings, holdout performance on a diagnostic criterion does not by itself validate identification, causal interpretation, or post-selection inference. The holdout should therefore be understood as a practical guardrail against hidden adaptive search, not as a full cure.

G Formal statement of the search problem

This appendix restates the protocol of Section 2 as a search problem and records, in standard terms, the distinctions that the main text states in plain language.

Let C denote the written instruction contract and let $\mathcal{T}(C)$ denote the class of editable scripts admissible under C . Let the available data be partitioned as $D = (D^S, D^H)$, where D^S is the search sample and D^H is a post-search holdout reserved for external validation. Let $S(\tau; D^S)$ denote the scalar score returned by the immutable evaluator when script $\tau \in \mathcal{T}(C)$ is run on the search sample. Both $\mathcal{T}(C)$ and S are fixed ex ante by the researcher as part of

the search design. When lower scores are preferred, the idealized search problem is

$$\tau^* \in \arg \min_{\tau \in \mathcal{T}(C)} S(\tau; D^S).$$

In practice, the agent sequentially generates and evaluates a finite sequence of candidate scripts $\{\tau_0, \tau_1, \dots, \tau_K\}$, each informed by the outcomes of previous attempts, and returns

$$\hat{\tau}_K \in \arg \min_{\tau \in \{\tau_0, \tau_1, \dots, \tau_K\}} S(\tau; D^S).$$

The distinction matters: τ^* is the best admissible script in the full class $\mathcal{T}(C)$, while $\hat{\tau}_K$ is the best script found by one bounded, path-dependent search. The audit log records the outcome of every experiment as an ordered sequence $L_K = ((k, \tau_k, s_k, d_k))_{k=0}^K$, where $s_k = S(\tau_k; D^S)$ when the run succeeds, $s_k = +\infty$ when it crashes, and $d_k \in \{\text{keep}, \text{discard}, \text{crash}\}$ is the outcome status defined in Section 2.1. Once the search is complete, the researcher may evaluate the selected script on the holdout, $S^H(\hat{\tau}_K; D^H)$, where S^H is a scoring function computed on D^H ; nothing in the loop requires D^H to exist during the search.

Two distinctions used in the main text can now be made precise. First, a restriction in the contract is *machine checkable* if a violation can be detected mechanically, without judgment. The evaluator enforces the information-set restriction by controlling which data are passed to the script. Other restrictions, including the editable file, runtime limit, and experiment budget, are mechanically auditable from the run logs, git history, and run counts, rather than actively prevented in this implementation. A restriction such as “do not combine peLASSO with other methods” is not machine checkable: any syntactically valid forecasting script runs and receives a score, whatever method it implements. Second, the *effective admissible set* $\mathcal{T}(C)$ is the set of scripts that satisfies the machine checkable restrictions. It may be strictly larger than the set of scripts the researcher had in mind when writing the contract, which we call the researcher’s *semantic intent*. In conventional specification search, the admissible set is an explicit menu written down in advance, so the effective set and the semantic intent

coincide by construction; in agentic search over programs, they can diverge. Drift, as used in Section 3.3, is a search path that remains inside the effective admissible set while leaving the semantic intent. The protocol makes containment in the first set verifiable and discloses the entire path; it does not, and cannot, guarantee containment in the second. The post-search holdout is the instrument that evaluates whether a departure from the semantic intent generalizes beyond the search sample.