AI Autoresearch for Economists

A Tutorial with Three Applications

Minchul Shin*

Current version: August 23, 2026
First version: July 2026

Abstract

We apply an automatic research workflow built around an AI coding agent to a wide range of economic research problems. Specifically, we apply the workflow to replication materials from three articles in the *Journal of Applied Econometrics*. In a global-inflation application based on Gillitzer and McCarthy (2019), the agent achieves an orders-of-magnitude computational speedup while preserving the original outputs. In a parental-leave application based on Troccoli (2024), an adversarial search across outcomes, subgroups, controls, and uncertainty methods asks whether any cell in a declared multiverse can overturn the paper's conclusion. The complete family provides no statistically persuasive evidence. In a local-projection application based on Piger and Stockwell (2025), the agent develops a horizon-regularized estimator with a relative mean-squared error of about 0.70—roughly 30 percent lower than that of the better of two reference estimators. Across the three applications, the workflow records the objective, the editable code, every attempted path, failures, and post-search validation. We also provide a minimal starter kit that economists can adapt to their own research problems.

Keywords: AI agents, autoresearch, empirical economics, research transparency, computational reproducibility, specification search

*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.

1 Introduction

Empirical researchers rarely run only one specification. They change controls, samples, estimators, numerical algorithms, and tuning parameters, learning from each result. AI coding agents make this process cheap and fast. The change is not only speed. A researcher can give an agent a codebase and a score; the agent can edit the code, run it, and choose the next experiment from the result. This creates an adaptive loop that resembles how economists conduct research: propose a change, evaluate the result, and use what was learned to choose the next step.

This type of loop has potentially wide applicability in empirical economics. Its central requirement is a well-defined, verifiable score. Depending on the question, the score might measure computational time subject to output equivalence, out-of-sample mean-squared error, the finite-sample efficiency of an estimator for a population target, or, in a deliberately adversarial exercise, the strength of statistical evidence within a prespecified class of specifications. Once the scoring rule and scientific constraints are fixed, an AI coding agent can enter the loop and autonomously explore ways to improve the score.

Shin (2026) adapts Karpathy (2026)'s `autoresearch` idea to empirical economics. That paper studies a forecast-combination problem from Diebold and Shin (2019) and ultimately identifies a method that improves out-of-sample predictive performance by exploiting serial correlation in forecast errors, a feature that the original study neither discusses nor uses. This example provides a promising result, but forecast combination represents a narrow empirical setting.

This tutorial takes the next step. We treat the workflow as a general research interface and show how to use it for routine empirical tasks. We select replication materials from three articles in the *Journal of Applied Econometrics* and give the agent three different goals:

1. **Computational efficiency:** make an existing empirical calculation faster without changing any substantive output;
2. **Statistical significance:** search an adversarial outcome–subgroup–specification multiverse for the strongest challenge to an empirical conclusion, then expose the complete family through a

multiplicity-aware analysis; and

3. **Statistical efficiency:** search for a lower-MSE local-projection estimator under a calibrated data-generating process, then test the selected estimator on fresh simulations.

We use the three applications to demonstrate the workflow. In each one, the economist writes the economic question, score, data access, and hard constraints before the loop begins. The search agent modifies one research file. A separate evaluator checks each candidate and computes its score, while a ledger records every attempt. After the loop, the economist's main agent checks the audit trail, reproduces the selected result, and runs validation suited to the objective.

Section 2 introduces the four-file system and explains how the search is recorded and audited. Section 3 presents the post-search audit reports for the three applications, and Section 4 concludes. The Appendix gives a step-by-step adoption guide for economists who have not run a coding agent before.

2 Autoresearch: the four-file system

2.1 The original four-file interface

Karpathy's original `autoresearch` repository asks a coding agent to improve a language model under a fixed training-time budget (Karpathy, 2026). The agent repeatedly edits `train.py`, runs a fixed evaluation supplied by `prepare.py`, and records the outcome in `results.tsv`; the behavioral instructions live in `program.md`. The design is powerful because the research surface is deliberately small. The agent has substantial freedom inside the training file. The contract holds the evaluation fixed, and the post-run audit detects any later change.

We retain that interface and reinterpret its contents:

`program.md`: the contract.

A plain-language statement of the question, score direction, admissible ideas, scientific constraints, evaluator command, and experiment budget. Roughly speaking, this file can be

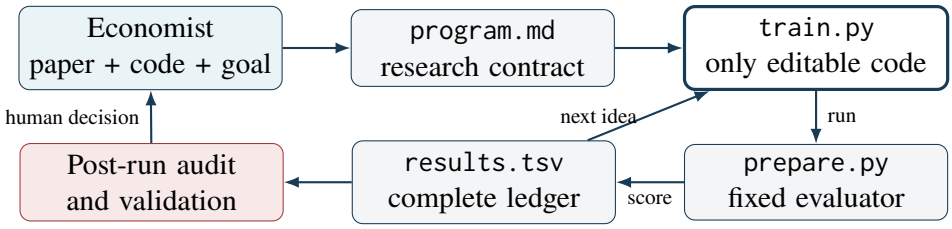

Figure 1: The four-file autoresearch interface

Notes: The search agent operates inside the edit–evaluate–log loop. The economist's main agent constructs the interface, presents the design for approval, and performs the post-run audit. Support files preserve raw event logs and reports outside the editable research surface.

viewed as a research-specific system prompt for the AI agent.

`train.py` (`train.*`): the editable research code.

The contract names this as the only research implementation the search agent may modify. It may contain an estimator, a forecast routine, or a specification selector.

`prepare.py` (`prepare.*`): the evaluator.

Fixed code that loads data, checks scientific constraints, and returns one scalar score together with diagnostics. A candidate that violates a hard constraint receives no usable score.

`results.tsv` (`results.*`): the audit ledger.

One ordered row for the baseline and every attempted candidate, including crashes, invalid outputs, duplicates, and timeouts.

2.2 A bounded adaptive search

In `program.md`, the agent is instructed to modify `train.py` in an effort to improve the score relative to earlier attempts. After coding a new idea in `train.py`, the agent runs the fixed evaluator in `prepare.py` to obtain a new score. It then records the score and status in `results.tsv` and repeats the process for K candidate attempts. These instructions are written in natural language.

This natural-language loop can be formalized as the following bounded adaptive search. Let τ_0 be the baseline implementation in the train file, and let $Q(\tau; D^S)$ be the scalar score returned by the

evaluator on search material D^S. For a minimization problem, the agent observes the history

$$L_{k-1} = \{(j, \tau_j, Q_j, \text{status}_j) : j = 0, \ldots, k-1\}$$

and proposes τ_k. The evaluator checks the candidate and, when it is valid, returns $Q_k = Q(\tau_k; D^S)$. After K numbered proposals, the selected implementation is

$$\widehat{\tau}_K \in \arg \min_{\tau \in \{\tau_0, \ldots, \tau_K\}:\text{valid}} Q(\tau; D^S).$$

The maximization case is analogous. The procedure conducts a bounded, path-dependent search rather than an exhaustive optimization. A second run may take a different path from the same starting point. The ledger records the path that the agent actually took.

In our examples, $K = 50$. We ask the agent to use the full budget even when an early proposal improves the score. Later attempts show whether nearby ideas fail, whether an apparent breakthrough survives small changes, and whether the agent has begun to repeat itself.

2.3 Recording the search

The `results.tsv` file provides a useful overview of the search. Each row records an attempt number, status, score, short description, and code hash. The ledger therefore shows the order of ideas and the resulting score trajectory. By itself, however, it may not contain enough detail for a full audit. A hash identifies a particular code file but cannot reconstruct it, and a short row does not necessarily reveal the exact edit, command, evaluator output, or failure. At least three recording approaches are available.

1. **Preserve the coding-agent log.** Coding-agent interfaces typically retain a transcript or event log containing observable edits, commands, outputs, and failures. Inspecting this record can reveal what the agent actually did at each step. When the interface exposes an exportable transcript or event stream, preserve that raw record under `.audit/raw/`. Because such logs can contain local

paths or sensitive output, they may need to be sanitized before they are shared.

2. **Use Git history.** Karpathy's original `program.md` creates a dedicated experiment branch and instructs the agent to commit each candidate before running it (Karpathy, 2026). The corresponding short commit hash is recorded in `results.tsv`. Improving commits remain on the branch, while non-improving candidates are reset. If rejected candidates must remain reproducible indefinitely, their commits should receive durable references, such as per-attempt tags, rather than being left for eventual Git garbage collection.

3. **Write a detailed record for every attempt.** The contract can instruct the agent to save an exact train-file snapshot, evaluator output, error log, and fuller explanation after every loop. For example, these records can be stored under `.audit/attempts/001/`, `.audit/attempts/002/`, and so on. This approach is explicit and does not depend on a particular coding-agent interface or version-control workflow.

Our current `program.md` template does not require per-attempt Git commits or detailed snapshots beyond the rows in `results.tsv`. In our three applications, we invoked the coding agent with structured event output and preserved that observable log for the post-search audits. This was a recording choice rather than a feature of the four-file interface. Session histories are commonly available in coding-agent interfaces, but their retention is not universal. Users should therefore verify that the selected interface preserves its log or add an explicit Git or per-attempt recording rule to `program.md`.

2.4 The post-run audit

After the search, the economist audits whether the run followed the contract and whether the selected result holds up. The audit asks:

- Were attempts 1 through K actually evaluated, and were failures kept?
- Did the contract, evaluator, data, sample, score, or tolerances change?
- Does the selected file reproduce its reported score?

- Did the candidate achieve the stated goal by changing something the economist meant to hold fixed?
- What broad families of ideas worked, failed, or exposed a weakness in the objective?
- What new evidence, not used during the search, can test the result?

The first-pass audit can itself be automated with an AI coding assistant. For example, the economist can prompt:

> We have just completed the research experiment described in `program.md`. Inspect the contract, evaluator, `results.tsv`, final train file, and preserved coding-agent log or Git history. Report what was attempted, what succeeded or failed, whether the selected result reproduces, and what limitations remain. Write the audit report in LaTeX.

The requested output can instead be HTML or Markdown, depending on the user's preference. The audit can rely on the coding-agent log or on Git commits when `program.md` specifies a Git recording rule. Automating the review does not remove the need for the economist to interpret the evidence. The economist sees one clearly described design before launch and receives one report after the declared budget. That report turns crashes, repeated ideas, changed files, and weak validation into concrete choices: accept the code, revise the contract, inspect a failure, or rerun.

Section 3 shows what such reports look like in practice. Its three application subsections are minimally edited versions of post-search audit reports written by AI coding agents from the preserved search records. They give readers a sense of the analysis a user can receive after a run.

2.5 How the same files represent different research goals

Table 1 maps the generic interface to the three applications. The editable file is always narrow, but it need not contain the same kind of object. In the first example it is an algorithm, in the second a specification choice, and in the third an estimator.

Table 1: Mapping the four files to three economic applications

File or decision	**Example 1** (Gillitzer and McCarthy, 2019)	**Example 2** (Troccoli, 2024)	**Example 3** (Piger and Stockwell, 2025)
`program.md`	Minimize time while preserving 150 forecast outputs.	Maximize the signed treatment statistic within a frozen 90-cell multiverse.	Minimize relative integrated MSE against two published LP estimators.
`train.py`	`compute(levels)`: the complete forecast calculation.	`choose_specification()`: four labels selecting one admissible cell.	`estimate_irf(system, shock)`: two outcomes over 37 horizons.
`prepare.py`	Checks hashes, output keys, 10^{-10} equivalence, determinism, and runtime.	Loads PISA data, estimates the selected cell, checks sample size and grid membership, and reports coefficient, SE, and t.	Evaluates candidates against known simulation truth and checks shape, determinism, sample response, and shock-scale equivariance.
`results.tsv`	Seconds, validity, implementation idea, hash, and failure status.	Signed t, selected cell, validity, hash, and failure status.	Relative MSE, diagnostics, estimator idea, hash, and failure status.
Post-search check	Batched timing, exact-output reproduction, and perturbed inputs.	The complete 90-cell family with raw and multiplicity-adjusted inference.	512 fresh simulations and a descriptive empirical application.

3 Post-search audit reports for three applications

We chose three papers with replication materials from a *Journal of Applied Econometrics* replication databank. Together they cover macroeconomic forecasting, a microeconomic causal application, and macroeconometric estimator design. For each paper, we reconstruct one empirical task and attach a new, explicit objective.

All three searches used the `gpt-5.6-sol` coding-agent model and the ordinary workspace-write environment. The runtime illustration used medium reasoning; the two statistical searches used xhigh reasoning. We report these settings because the model and reasoning effort can affect the search path. Each run used 50 candidate attempts, followed by an audit outside the search loop.

After each run, we asked the coding agent to write a post-search audit analysis based on the search record. The following subsections reproduce those agent-written reports with minimal editing. They give readers a sense of the post-search analysis that an AI coding agent can provide to the user.

3.1 Example 1: Computational speed: global-inflation forecasts

Original question and research design

Gillitzer and McCarthy (2019) ask whether information in global inflation improves inflation forecasts for industrialized economies. Their global-information forecast augments a domestic benchmark with the gap between global and domestic inflation. The replication calculation covers 23 economies, forecast horizons of four and eight quarters, and full-sample, rolling-sample, and grid-selected forecast combinations. Our baseline returns 150 country and aggregate values matching the preserved replication result.

Why make this an autoresearch problem?

Replication code often demonstrates an analysis once. Later work may need to run the same calculation thousands of times in simulations, robustness checks, or classroom exercises. We therefore ask the agent to minimize the median wall-clock time of `compute(levels)` while treating

output equivalence as a hard constraint. The evaluator requires the same 150 keys and numerical agreement within 10^{-10}. It rejects skipped calculations, embedded answers, caches keyed to the benchmark data, and weakened tolerances regardless of speed.

What the loop found

The baseline took 2.217 seconds. Attempt 1 replaced pandas-heavy loops with NumPy arrays and cumulative regression sufficient statistics, reducing the reported time to 0.00185 seconds while retaining every output. The next experiments simplified date handling. Attempt 7 used the quadratic form of the unchanged forecast loss to identify the minimizing point on the original 101-point weight grid, rather than recomputing every loss from scratch. Later attempts shortened arrays to the evaluation window, used contiguous slices, and batched output assembly. Attempt 45 has the best reproducible score and becomes the selected implementation.

The complete ledger contains 46 valid candidates, one error, and three duplicates. Attempt 12 tried an in-place logarithm on a read-only array and failed; the failure remains visible. The later duplicates show that the agent had reached a narrow region in which timing noise could reorder sub-millisecond variants.

Fresh evaluation reproduced all 150 outputs with maximum absolute error 5.77×10^{-15}. Five baseline calls and 5,000 selected calls produced median times of 2.174083 and 0.000238583 seconds, a ratio of about 9,112. Hardware and timing design affect that ratio; the original three-call evaluator cannot reliably rank the final sub-millisecond variants. The audit supports a structural, orders-of-magnitude speedup with unchanged economic output.

3.2 Example 2: Red-teaming a paper-wide conclusion: paid parental leave

Original question and research design

Troccoli (2024) revisits estimates of how an Austrian paid-parental-leave extension affected children's later PISA scores. The design compares children born before and after the July 1990 eligibility

cutoff and uses an older cohort to account for month-of-birth patterns. In a simplified notation, the coefficient of interest is the interaction in

$$y_i = \alpha + \beta_1 \text{PostJune}_i + \beta_2 \text{Cohort1990}_i + \beta_3 (\text{PostJune}_i \times \text{Cohort1990}_i) + \text{BirthMonth}_i' \theta + X_i' \mu + \varepsilon_i.$$

PISA reports five plausible values for each outcome and 80 balanced replicate weights for its complex sampling design. Danzer and Lavy (2018) use only the first plausible value (PV1) and student weights. For boys with highly educated mothers, they report a science coefficient of 40.40 with a standard error of 11.45 ($N = 482$). Troccoli first reproduces that PV1 specification almost exactly: 40.38 with a standard error of 11.45 ($N = 484$). The two-observation sample difference therefore does not explain the change in results.

Troccoli then combines all five plausible values. The science coefficient for the same subgroup falls to 21.40, with a Rubin standard error of 21.38. Adding the replicate weights leaves the coefficient unchanged and produces a BRR standard error of 21.48. Averaging the five plausible values reduces the large PV1 point estimate, while Rubin's rules and BRR incorporate imputation and survey-design uncertainty. Troccoli's Table 1 reports the same sequence for mathematics, reading, and science in the full sample and four gender-by-maternal-education subgroups. These published outcome–sample cells define the starting family for our search.

Why use an adversarial objective?

We target Troccoli's paper-wide conclusion rather than one coefficient. We let the agent search for the strongest positive evidence across every published outcome and subgroup, three nested control sets, and two uncertainty methods. The search therefore changes the outcome and sample as well as regression and inference choices. It asks whether any cell in the declared family challenges the conclusion that appropriate treatment of plausible values and sampling uncertainty removes the original headline significance. This is an adversarial outcome–subgroup–specification multiverse, not a robustness check around one fixed estimand.

RED-TEAM / CAUTIONARY. We maximize the signed positive t-statistic for the parental-leave coefficient. The frozen multiverse contains 90 cells: three outcomes, five samples, three control sets, and two uncertainty methods. The editable file chooses one cell; the evaluator loads the data, enforces the literal grid and 400-observation minimum, and estimates the model. We place the winning statistic in the complete family, report multiplicity-adjusted inference, and do not treat the selected cell as confirmatory evidence.

For a narrower and often more realistic stress test, we would fix one published outcome and vary the remaining analytic choices. For example, we could fix science and let the agent vary the sample, controls, and uncertainty method. An even tighter design would also fix the target subgroup and vary only regression and inference choices. The experiment reported here instead asks the broader red-team question across the paper's full family of outcomes and subgroups.

What the stress test found

The baseline uses the first canonical cell in Troccoli's table: mathematics, the full sample, full controls, and Fay balanced repeated replication (BRR). Mathematics starts the ledger because it is the table's first outcome, not because the design gives it substantive priority. Attempts 1–15 screen the mathematics cells with Rubin uncertainty, attempts 16–30 screen reading, and attempts 31–45 screen science. Attempts 46–50 revisit the five leading cells with BRR. The trajectory follows this enumeration order; it does not represent evidence accumulating over time. The running-best score rises from 0.0119 at the baseline to 0.9618 for mathematics and then to 1.2237 for science. Attempt 36 selects science, boys with highly educated mothers, cohort-and-birth controls, and Rubin uncertainty.

Within that outcome–subgroup pair, the full-control Rubin estimate is 21.399 (standard error 21.375), reproducing Troccoli's reported 21.40. The search chooses leaner cohort-and-birth controls, dropping paternal-education, family-status, and school-location controls. With the outcome, subgroup, sample, and Rubin method fixed, the coefficient rises by 4.323 to 25.722, the standard error dips to 21.020, and the signed t-statistic rises from 1.001 to 1.224 (two-sided $p = 0.226$).

Re-estimating the selected cell with BRR gives standard error 21.245, $t = 1.211$, and $p = 0.230$. More importantly, the post-run agent evaluates all 90 cells. None has a two-sided raw p-value below 0.05; the smallest Holm-adjusted p-value is 1.000 and the smallest Benjamini–Hochberg q-value is 0.976.

The agent used all 50 attempts, stayed inside the specified family, and found the largest signed statistic. Favorable cells concentrate among boys with highly educated mothers, while several cells for boys with low maternal education are negative. The complete family shows no persuasive evidence, so Troccoli's conclusion survives this frozen 90-cell challenge. The result applies only to the declared multiverse; the selected cell supports no confirmatory causal or policy claim.

3.3 Example 3: Estimator development: local projections

Original question and research design

Piger and Stockwell (2025) ask whether local projections with an externally identified, observed shock should place the future outcome level or a cumulative long difference on the left-hand side. Levels and long differences target the same impulse response but can have different finite-sample behavior. The paper finds that long differencing can substantially reduce bias and improve interval coverage, although its variance can be higher.

The empirical application uses monthly U.S. industrial production and consumer prices with monetary-policy and central-bank-information shocks. Holding the shock, controls, sample, and horizons fixed, the levels and long-difference specifications produce notably different cumulative industrial-production responses over horizons 0–36. Our translation reproduces the paper's rounded values of approximately -1.3 and -8.6.

Why search for a third estimator?

Neither reference estimator uniformly dominates in mean-squared error. This suggests a more ambitious goal than reproducing their comparison: can an agent construct a direct LP-family point

estimator with lower finite-sample MSE than both?

We estimate a stable five-variable Gaussian VAR(12) from the empirical system and use it as a calibrated DGP. The true impulse responses are known in simulation. The search bank contains 32 fixed samples of 372 months. For each of four shock–outcome paths, the evaluator divides the candidate's integrated MSE over horizons 0–36 by the lower MSE of the two frozen references. The score is the mean of the four ratios, so a value below one means that the candidate beats the path-specific better reference on average.

The editable function receives only one simulated system and shock and must return two 37-horizon responses. It cannot read files, identify a replication, or access truth. The shock coefficient must be estimated directly from the supplied sample at every horizon. After selection, the search agent cannot see the fresh validation simulations or empirical results.

What the loop found

The baseline long-difference LP scores 1.6243. Early experiments reduce the number of nuisance lags and change whether controls enter in levels or differences. Attempts 15–29 add increasingly strong second-difference smoothing across horizons. That improves aggregate MSE but initially damages the short-horizon score: one aggressive smoother reaches a short-horizon relative MSE as high as 2.86. The diagnostic changes the direction of search. Attempts 30–41 explore hard boundaries and then extra fidelity weight near impact. Attempts 42–46 add future shocks as nuisance controls; they worsen performance, so the search discards them.

The search selects attempt 49. It uses a long-difference outcome, four lags of all five system controls in levels, an unpenalized current-shock coefficient estimated separately at every horizon, and a cross-horizon second-difference penalty of 100,000 with a 100-fold fidelity weight at horizon zero. Its search-bank score is 0.7011, 56.84 percent below the baseline. All 50 candidates are valid, 48 improve on the baseline, and 39 score below one.

The important result comes after search. With the estimator and tuning constants frozen, 512 simulations from a newly chosen seed produce a score of 0.6847. The candidate beats both references

for all four shock–outcome paths and for the short, medium, and long horizon groups. The gain is primarily variance reduction: the candidate has more integrated squared bias than either reference on the fresh simulations, but horizon pooling lowers variance enough to reduce MSE.

Table 2: Selected local-projection estimator: relative MSE

Metric	Search bank (32)	Fresh simulations (512)
Mean relative MSE	0.7011	0.6847
Monetary policy → industrial production	0.7109	0.7457
Monetary policy → CPI	0.9481	0.7194
Central-bank information → industrial production	0.6394	0.6544
Central-bank information → CPI	0.5059	0.6194
Short horizons, 0–6	0.4762	0.5499
Medium horizons, 7–18	0.4632	0.4919
Long horizons, 19–36	0.7348	0.7715

Notes: Each path ratio compares the candidate MSE with the lower MSE of the levels and long-difference reference estimators. Lower is better.

Applied to the original empirical sample, the selected estimator produces cumulative responses over horizons 0–36 of -2.30 for monetary policy to industrial production, -1.45 for monetary policy to CPI, -3.90 for central-bank information to industrial production, and 0.20 for central-bank information to CPI. We report these values as a descriptive application because the real causal response is unobserved. The selected smoothed estimator also requires its own inference procedure; the paper's Newey–West intervals do not transfer automatically. Estimator-specific coverage is the next research step.

3.4 Reading the three audit trails together

Figure 2 plots each candidate and the running best. The vertical axes have different interpretations. A falling runtime curve is good only after output equivalence passes. A rising stress-test t-statistic

means that the adversary found a more favorable cell; only the complete family reveals whether that challenge overturns the empirical conclusion. A falling simulated MSE is useful evidence about estimator performance under the calibrated DGP. It does not reveal whether the selected empirical curve is close to the unobserved truth.

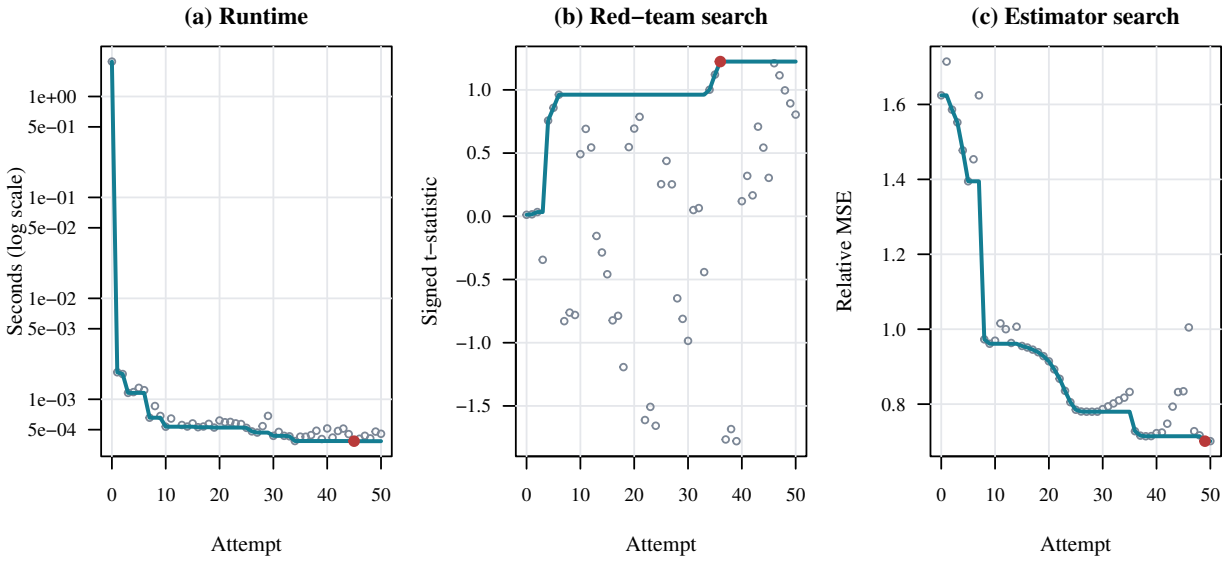

Figure 2: Search paths across the three applications

Notes: Open circles are evaluated candidates, the teal line is the running best, and the red point is the selected attempt. Panel (a) is minimized and uses a log scale; panel (b) is maximized but is a cautionary objective; panel (c) is minimized. The full ledgers include idea descriptions, statuses, elapsed times, and code hashes.

Table 3: Run and audit summary

Application	Budget used	Valid	Other outcomes	Selected result
Global-inflation runtime	50/50	46	1 error, 3 duplicates	Exact 150 outputs; large reproduced speedup; precise ratio remains timing-sensitive.
Parental-leave stress test	50/50	50	None	$t = 1.224$, but $p = 0.226$, Holm $p = 1.000$, and BH $q = 0.976$.
Local-projection estimator	50/50	50	None	Relative MSE 0.701 on search and 0.685 on 512 fresh simulations.

4 Conclusion

Autoresearch gives the economist a simple way to organize adaptive code search. The economist chooses the question and incentive. A coding agent proposes and tests changes. Four files separate the contract, editable implementation, evaluation, and memory. The post-run audit brings the evidence back to the economist.

Across three JAE applications, the workflow behaves differently because the objectives differ. It finds a structural speedup while preserving empirical outputs. It uses an adversarial outcome–subgroup–specification search to ask whether any cell in a frozen multiverse can overturn a paper-wide conclusion and finds no convincing challenge. It constructs a horizon-regularized local-projection estimator that survives fresh simulations under a calibrated DGP. In every case, the audit trail is more informative than the winning file alone.

AI models and the software harnesses around them are improving rapidly. The natural-language research loop demonstrated in this tutorial, for example, would have been difficult to implement

reliably with the coding-agent systems available in early 2025. As model and harness capabilities continue to improve, specification search and other parts of empirical research are likely to become increasingly automated. Designing efficient, transparent, and responsible workflows for AI research agents in economics is therefore an important direction for future work.

Appendix

A A practical adoption guide

A.1 Start with a paper, code, and goal

Download the companion GitHub repository as a ZIP file or clone it, then enter the `autoresearch-starter-kit` folder. Open Codex, Claude Code, OpenCode, or another coding agent from that folder. The repository provides the minimal structure and instructions needed to begin immediately with an ordinary agent conversation.

Before starting, place three ingredients in the workspace:

1. your paper or draft in a format the agent can read, preferably TEX, Markdown, or plain text;
2. executable code that produces the numbers, tables, figures, or other parts of the paper you want to work on; and
3. a goal with a measurable direction, such as reducing runtime or out-of-sample RMSE, improving statistical efficiency, or minimizing a p-value in an explicitly labeled red-team exercise.

Copy the paper and executable code into the repository, typically under `projects/my-study/input`, together with any data you need. The goal need not be final. If it is unclear, the next step shows how to develop it with the agent; the paper and runnable code should already be available for the agent to inspect.

A.2 Define the experiment with a coding agent

Once the agent can see the paper and code, describe what you want it to improve. A first prompt can be as short as:

> My paper and replication code are in `projects/my-study/input`. Convert them into the four-file autoresearch system in this repository. I want to improve [state the goal].

If the goal is uncertain, use:

> Read the paper and code in `projects/my-study/input`. Suggest three useful autoresearch goals, explain how each would be scored, and ask me which one to use.

Before the coding agent launches the search, ask it to explain:

- what function or script the search agent will be allowed to change;
- the score and whether lower or higher is better;
- which sample, estimand, output, or tolerance remains fixed;
- the baseline score and approximate evaluation time; and
- the experiment budget, defaulting to 50.

The coding agent first runs the baseline and estimates the total runtime. Once you approve the design, it launches the search and manages the agent sessions.

A.3 Inspect the generated four files

The agent creates:

```
projects/my-study/autoresearch/
  program.md
  train.py       # or train.R
  prepare.py     # or prepare.R
  results.tsv
```

Before approving the run, make sure you can answer four questions without reading all the code:

1. What number is the agent trying to improve?
2. What would count as cheating or changing the scientific question?
3. Which single file may the search agent modify?

4. What evidence will be reserved until after selection?

If those answers are unclear, revise `program.md` or the evaluator before spending the experiment budget.

A.4 Start the loop, then audit it

Once you approve the research contract, send the coding agent—still opened at the starter-kit root—a short prompt that identifies the experiment:

> Read `projects/my-study/autoresearch/program.md` and kick off the loop.

This is enough because the named `program.md` already states the editable file, evaluator command, score, constraints, logging rule, and number of experiments. The agent follows that contract through the declared budget and records each attempt in `results.tsv`.

Your coding agent preserves the raw event stream. When the search exits, it produces `audit/findings.json`, `audit/report.tex`, and, when a TeX compiler is available, `audit/report.pdf`. Use the report to check:

- how many ideas the agent attempted and how many failed, timed out, or repeated;
- whether the run changed files outside the permitted surface;
- whether the selected score reproduces;
- which families of ideas worked and failed;
- whether the apparent gain exploits a weakness in the score; and
- what post-search validation says.

You then decide whether to keep the implementation, inspect it, change the contract and rerun, seek new validation, or abandon the objective. A better search score starts that decision; it does not settle it.

B Compact checklist for a new application

1. Preserve the supplied paper, code, and data.
2. Identify one executable function connected to the proposed goal.
3. Define a scalar score and hard scientific constraints.
4. Create the four files and run the baseline once.
5. Estimate runtime as the evaluator time times the budget, plus allowance for agent reasoning.
6. Explain that the selected coding service will receive the relevant project contents, and obtain approval.
7. Run all numbered attempts and retain the raw log.
8. Reproduce the winner and inspect changed files.
9. Use post-search evidence the agent could not optimize.
10. Report the path, limitations, and human decision – not only the winner.

References

Natalia Danzer and Victor Lavy. Paid parental leave and children's schooling outcomes. *The Economic Journal*, 128(608):81–117, 2018. doi: 10.1111/ecoj.12493.

Francis X. Diebold 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.

Christian Gillitzer and Martin McCarthy. Does global inflation help forecast inflation in industrialized countries? *Journal of Applied Econometrics*, 34(5):850–857, 2019. doi: 10.1002/jae.2704.

Andrej Karpathy. autoresearch. GitHub repository, 2026. `https://github.com/karpathy/autoresearch`.

Jeremy Piger and Thomas Stockwell. Differences from differencing: Should local projections with observed shocks be estimated in levels or differences? *Journal of Applied Econometrics*, 40(7):759–787, 2025. doi: 10.1002/jae.70003.

Minchul Shin. An auditable AI agent loop for empirical economics: A case study in forecast combination. *Economics Letters*, 2026. doi: 10.48550/arXiv.2603.17381. Accepted for publication.

Claudia Troccoli. Does paid parental leave affect children's schooling outcomes? replicating Danzer and Lavy (2018). *Journal of Applied Econometrics*, 39(2):356–362, 2024. doi: 10.1002/jae.3021.