Rewriting the Past: How an AI-Assisted Benefits Model Handles Dated Rules

Shirshak Jung Thapa, Florida Atlantic University, Boca Raton, Florida, United States. *Preprint, not peer reviewed, November 2026.* *License: CC BY 4.0.* *DOI: https://doi.org/10.5281/zenodo.24406653*

Abstract

Benefit rules change on set dates, but hearings, corrections and reviews concern earlier months, so software encoding the rules must still answer correctly for the past. We studied how one open-source policy model, PolicyEngine-US, handled its dated rules for the Supplemental Nutrition Assistance Program (SNAP) during 2026, a year in which 89 of the 101 commits that touched a dated value named an AI tool as co-author. Comparing every dated entry before and after each commit, we find that revising the past is routine. Seventy-two commits changed a period already in effect, new periods were entered a median of 157 days after they took effect, and 29 of 33 fixes repaired entries written earlier the same year. Two commits altered correct entries: one by accident, one by trusting a faulty official table. In a controlled test, an AI coding agent put ten real changes back into PolicyEngine-US from short notices. Across 20 runs it damaged no result or entry for earlier months and made both required corrections to the past. Its three errors concerned fit with the existing code; two were caught only by the maintainers' tests. One formula reads the rules at a fixed date and bypasses the dated record, denying an exemption to adults aged 51 to 54 in months when the law granted it. Wrong answers about the past came from late entries, repeated corrections and code that bypasses the record, not from an agent overwriting history. The study covers one codebase, one program and one year.

Keywords: Rules as code; public benefits; SNAP; AI coding agents; software maintenance; temporal data; repository mining

1. Introduction

On 6 March 2026, a change was recorded in PolicyEngine-US, an open-source model of United States tax and benefit rules, which we call the policy model. Its purpose was to add old utility allowances for every state, so that the policy model could work out food assistance for earlier years. In Maryland it did something more. It altered four entries that were already correct. The allowance for 2023 went from $505 a month to $551, which was the following year's figure. Two start dates moved from January to October. The entry for 2025 was deleted. Three days later a second change put all four back, citing Maryland's own notices. In between, the policy model used a different allowance for a Maryland household in 2023 than it had the week before, and nothing in Maryland's rules had changed.

Benefit rules change on set dates, and questions about earlier months keep arriving after those dates have passed. A household asks for a hearing on a decision made in the last 90 days. A state restores benefits it wrongly withheld up to a year earlier, or recovers an overpayment from as much as six years back. A reviewer rechecks a sampled case. An analyst compares what a program paid in 2023 with what the rules said it should pay (7 C.F.R. §§ 273.15, 273.17, 273.18, 2024; 7 C.F.R. pt. 275, 2024). A policy model that writes the rules as computer code, an approach known as rules as code, has to answer each of these with the rules as they stood in that month. It does so by storing every value together with the date it took effect. The record of

the past is therefore stored inside the program itself, so any edit to the program, even one meant only to add something new, can also change what it says about earlier months.

In 2026 much of that editing was done with an AI coding tool. The code of PolicyEngine-US is kept in a public repository, where each recorded change is called a commit. That year, 101 commits added, changed or removed a dated value for the Supplemental Nutrition Assistance Program (SNAP), the main federal food assistance program. Of those, 89 carry a line naming Claude, an AI model, as co-author. To keep the two apart, we write "policy model" for PolicyEngine-US and "AI model" for Claude. This makes the repository an unusual public view of AI-assisted upkeep of benefit rules, at a time when researchers and public-interest technology groups are testing AI tools for writing benefit rules as code (Kennan et al., 2025). Studies so far have asked AI tools to write new rules from the law; we found none that measures what keeping a policy model up to date does to its rules for earlier months, whether the work is done by people or with AI tools (Section 7).

We ask two questions, answer them with a study of one year of changes and a controlled test, and follow one finding with a check of every SNAP formula. The first is what a year of such upkeep does to the dated record: how often the past is changed, for what reason, and how often the change is itself wrong. The second is the worry that the Maryland case raises. When an AI coding agent, a tool that reads the code, edits files and runs tests without step-by-step direction, is asked to add one dated rule change, does it damage the months before that change?

Both questions matter beyond this one codebase. Policy models like this one are built to estimate what households may receive and to analyze the effects of rules, including in earlier years, and every use listed above depends on the rules for a past month being right. If upkeep quietly changes those rules, nothing looks wrong today. The error appears later, in an answer about an old case or an analysis of an earlier year, where nobody is looking for it. As AI tools take on more of this upkeep, the people who maintain policy models like PolicyEngine-US, and the organizations deciding whether to let AI tools help with them, need to know whether edits made with AI tools put the rules for past months at risk, and where checking is worth the effort. Evaluations of AI coding agents mostly ask whether a change works now (Y. Wang et al., 2026); this paper asks whether it leaves earlier months intact, and what goes wrong with them instead.

Two limits apply to what the paper says. Most commits name Claude as a co-author. That line tells us only that the AI tool was used in making the commit. It does not tell us which parts the tool wrote and which the people wrote, who chose each value, or who checked it. So when we find an error, we report what the commit changed, and we do not attribute the error to the AI tool or to the people who used it. And PolicyEngine-US is an independent calculator: this paper studies only its code, not the computer systems that state agencies use to decide who receives benefits, and its findings say nothing about them.

To answer the first question, we looked at the numbers PolicyEngine-US uses to calculate SNAP benefits, such as income limits, deductions and each state's utility allowance, each stored with the date it takes effect. We compared every one of them before and after each of the year's 101 commits, and recorded every number that was added, changed or deleted: 4,960 in all. Most of the work turned out to be on the past. Seventy-two commits changed a value, a start date or an entry for months that had already begun. Only four added a new rule without touching any earlier month. New rules also tended to arrive late: of 736 new entries, 616 were added after the date they took effect, a median of 157 days afterwards. We then read each of the 72 commits that changed the past, to see why. Thirty-three fixed a wrong value or a wrong start date, and 29 of those

fixed entries that had themselves been added earlier in 2026. Two commits altered entries that were already correct: the Maryland change, by accident, and a correction that brought in 14 wrong values from a faulty official table and was reversed within a week. In other words, keeping the policy model right meant going back over its past again and again, mostly to repair recent work.

For the second question, we ran a controlled test. We took ten real changes from 2026, removed each one from a copy of PolicyEngine-US that had no history to look it up in, and asked an AI coding agent (Claude Code, using the AI model Claude Sonnet 5.5) to put it back from a short description we wrote. Each task was run twice, once with an example household it could test its work on and once without. Two of the ten tasks were corrections, in which the right answer was to change earlier months. "Damaging the past" has a precise meaning here: a household's benefit changes for a month before the new rule starts, or a stored number for such a month is altered. In 20 runs, that never happened, and both corrections were made correctly. The agent made three other errors: it put a rule in the wrong part of the code, added an exemption to one list but not to a second, and created a duplicate of an input that already existed. Two of the three still gave the right benefit for every household we tried, and only the maintainers' own tests, which the agent could not see, caught them.

Preparing one of the test tasks, we found a different kind of error. Adults without dependents must meet a work requirement to keep receiving SNAP beyond three months, unless they are old enough to be exempt. From 1997 the exemption began at age 50; a 2023 law raised that age in steps to 55. PolicyEngine-US stores these steps correctly, each with its date. But the formula that decides who is exempt (the piece of code that does this one calculation) does not use them. Since February 2026 it has read the rules as they stood on 1 June 2025, when the age was already 55, and applied them to every earlier month. So a 52-year-old in Texas in June 2022, exempt under the law then in force, is treated as not exempt, and PolicyEngine-US computes no benefit for them. Three of the maintainers' tests expect this answer, so the tests pass. We then searched all 115 SNAP formulas for others that read the rules at a set date. We found three more, and each gives the right answer for every month. Only the exemption formula gets the past wrong. The same error had been reported in the project's issue tracker in September 2026, and a correction was proposed in November; it had not been merged when we wrote (Section 5.1).

Put together, these results change the picture. The worry was that an AI agent asked to add a new rule would overwrite the rules for earlier months. In our test it never did. The wrong answers about the past came from elsewhere: new rules entered months after they took effect, values that took four passes to settle, one of which brought in errors from an official table, one large edit that reached entries it was not meant to touch, a formula that ignored the dates, and new rules that did not fit the code around them. And changing the past was not a rare accident but regular work, spread across 23 different weeks of the year.

This paper makes four contributions:

- **A measurement of AI-assisted upkeep of dated benefit rules.** For one year of SNAP rules in a public policy model, it shows how often the rules for past months were changed, why, and how late new rules arrived (4,960 recorded changes; all 72 commits that changed the past, read and labeled).
- **A controlled test of an AI coding agent on dated rule changes.** Ten real changes, put back by an agent in 20 runs, show whether adding a rule damages earlier months and what errors occur instead.

- **A map of where wrong answers about the past come from**, including a formula that bypasses the dated record, checked against the law and also reported independently in the project's issue tracker, and a search of all 115 SNAP formulas for the same pattern.
- **Three checks that anyone maintaining a policy model can automate**: compare the entries already in effect before and after each change, compare each formula's answers for earlier months with the dated record, and compare each entry's date with the day it was entered. Run over the year, they would have flagged both damaging commits, the formula error and every late entry, at the price of reading about three commits a week (Section 5.6). The data, labels and scripts are released so that others can repeat the study.

The results are meant to be used. Maintainers of PolicyEngine-US and of similar policy models can run the three checks on every change and complete the correction already proposed for the formula error (Section 5.1). Organizations weighing AI tools for rules as code learn where review effort is best spent: on the sources of error above, more than on the agent's edits. Researchers who evaluate AI coding agents get a task design that tests a property most evaluations leave out, whether a change leaves the past intact, and a released set of real tasks to build on. Through all three, the people who rely on these models' answers about earlier months are better served.

These findings come from one codebase, one program and one year. The commits were labeled by one reader, and the controlled test is 20 runs with one AI model. Section 2 explains how PolicyEngine-US stores dated rules. Section 3 reports the year of commits, Section 4 the controlled test and Section 5 the errors that did occur. Section 6 sets out the limits, Section 7 the related work, and Section 8 concludes. The data, the scripts and the label for each commit are released with the paper (see Data availability).

2. Background: how PolicyEngine-US keeps time

PolicyEngine-US stores each rule in two parts. The numbers, such as the maximum benefit for each household size, sit in plain text files called parameter files, each number with the date it took effect. The steps of the calculation, such as subtracting 30 percent of a household's net income from that maximum, are written as program code in separate files called formulas. When an amount changes, only the number file needs a new entry; the code stays the same. Here is one such list of numbers: Maryland's standard utility allowance (a fixed amount for heating and utility costs that a household may count instead of its actual bills when SNAP works out its benefit), in dollars a month, as PolicyEngine-US records it for three years:

```
MD:
  2021-01-01: 388
  2022-01-01: 431
  2023-01-01: 505
```

Each line is an entry: from this date, this value. A value stays in force until the next entry begins, so $431 applies to every month of 2022 and $505 to every month of 2023. We call one such list a series, and the time from one entry to the next a period. All the entries together make up what we call the dated record: PolicyEngine-US's memory of what the rules were, and when. At the version we studied, the SNAP part of PolicyEngine-US holds 7,659 entries in 1,386 series across 125 parameter files. The earliest is dated 1969, and the latest are forecasts for 2036.

Most SNAP amounts change on 1 October, when the federal fiscal year begins. Not all do. Maryland sets its utility allowances from 1 January (Maryland Department of Human Services, 2024), which is why the entries above carry January dates, and why an edit that assumes October is wrong for Maryland.

The logic lives in formulas, 115 files of them for SNAP. A formula works out one quantity for one month, such as whether a person is exempt from the work requirement. To do that it asks for the rules of that month, and PolicyEngine-US hands it every value as it stood on the month's first day. A formula written this way needs no knowledge of history. Asked about June 2022, it receives the 2022 values. Asked about June 2026, it receives the 2026 values.

Two consequences matter for this paper.

The first is that a start date decides the month. The 2025 budget law was signed on 4 July 2025 (Pub. L. No. 119-21, 2025). A value dated 4 July is not yet in force on 1 July, so the policy model first applies it in August. The same value dated 1 July applies in July. Three days in the record move a rule by a month, and the repository's history contains both choices for provisions of that law.

The second is that a formula can ignore the month it is asked about. Instead of taking the rules for that month, it can name a date and read the rules as they stood on that day, whatever month is being calculated. Sometimes this is right. Some state tax laws adopt federal rules as they stood on a set date: Arkansas follows the federal credit for child and dependent care as it stood on 2 January 2013, and Oklahoma the federal earned income credit rules of 2020 (Ark. Code Ann. § 26-51-502; Okla. Stat. tit. 68, § 2357.43). PolicyEngine-US follows such laws by reading the federal values at those dates. Otherwise it is a trap. A formula that names a date for any other reason uses the same rules for every month, so a correction to the dated record never reaches its result.

This leaves three ways for PolicyEngine-US to give a wrong answer about the past. First, an entry in the dated record can be wrong: a value, a start date, or an entry added or deleted where it should not be. Second, a formula can read the record at the wrong date, as just described. Third, a rule can be written with no date at all, inside a formula or in a list with a single start date, so that it applies to every month PolicyEngine-US covers, including months before the rule existed. Section 3 measures the first. Section 5 reports cases of the second and third.

PolicyEngine-US also has tests, 113 files of them for SNAP. Each test is a worked example written by the maintainers: a household, a month, and the answer PolicyEngine-US should give. If a later change alters that answer, the test fails. This protects right answers, but it also protects wrong ones: if a test's answer is wrong, fixing the code makes the test fail, so the error stays. This paper finds both.

All results refer to one version of PolicyEngine-US: commit 47c4e8720 (version 2.118.10) of 13 November 2026. "Correct" has a precise meaning here: unless we name a legal source, it means agreement with the maintainers' code at that version.

3. A year of commits

3.1 What we counted

Between 1 January and 13 November 2026, 241 commits changed at least one SNAP file in the repository. For each one, we took every SNAP parameter file it changed, compared the version before the commit with

the version after, and went through each series entry by entry. A script sorted every difference into one of five kinds:

- **New period.** An entry dated later than any the series already had, such as the allowances for the coming October.
- **Earlier period filled in.** An entry added before the series' latest date, such as allowances for years before the series used to begin.
- **Value changed.** A different value at a date that already existed, as when Maryland's 2023 allowance went from $505 to $551.
- **Start date moved.** One date removed and another added with the same value, as when two of Maryland's start dates moved from January to October.
- **Date deleted.** An entry removed with nothing in its place, as when Maryland's 2025 entry was deleted.

We call the last three *corrections*, because each alters something the record already said. The name describes what a change does, not whether it was right: the Maryland edit counts as a correction too. A change is to a period *already in effect* when that period had begun by the day of the commit. Changing the 2023 allowance in March 2026 is a change to the past; adding the allowance for the coming October is not. The comparison produced 4,960 recorded changes. Of these, 4,347 were to entries inside existing series; the other 613 created, removed or moved a whole series.

Every later number rests on this sorting, so we checked it a second way. Each commit comes with its diff: the plain list of lines it removed and added. Working from those lines alone, without our script's reading of the files, we confirmed 4,229 of the 4,248 changes whose value fits on one line. We read the other 19 by hand, and all were right. The remaining 99 changes, whose values are lists rather than single numbers, could not be checked this way.

3.2 Most commits changed the past

Table 1. One year of SNAP commits

Commits in 2026 that...	Count
changed a SNAP file	241
added, changed or removed a dated series	101
changed entries inside an existing series	79
changed a period already in effect	72
added a new period	15
added a new period and changed no period already in effect	4

Table 1 shows how the year's commits divide. Of the 241 commits that changed a SNAP file, 101 changed at least one dated value; the other 140 changed SNAP files without touching any dated value, for example formulas or tests. Of the 101, 72 changed a period already in effect. Only 15 added a new period, and 11 of those also changed the past. In the whole year, just four commits added a new period without changing any period already in effect.

Table 2. What happened to dated entries

Change	Entries	Already in effect
New period	736	616
Earlier period filled in	3,049	3,049
Value changed	288	270
Start date moved	190	190
Date deleted	84	84

Table 1 counts commits; Table 2 counts the entries they changed. The largest group, 3,049 entries from 21 commits, filled in earlier periods. These commits extended PolicyEngine-US backwards, mostly with utility allowances for earlier years, so that it could calculate benefits for those years. That is the past being written for the first time. The 562 corrections (a value changed, a start date moved or a date deleted) are the past being rewritten, and 544 of them fell on periods already in effect. The 736 new periods are the subject of the next subsection.

3.3 New periods arrive late

Of the 736 new periods, 616 were entered after they had already taken effect, a median of 157 days late. One event accounts for most of them. The utility allowances for fiscal year 2026, which took effect on 1 October 2025, were entered on 7 March 2026, 157 days later: 443 entries on a single day. That one day is why the median is exactly 157 days. It does not explain the lateness away. Of the 173 late entries made on other days, 157 came from the child support correction of 24 September (Section 3.6), which added entries for 2022 to 2024, and the other 16 were a median of 137 days late. Counted by commit, 12 of the 15 commits that added a new period added at least one that was already in effect.

A late entry produces wrong answers without any faulty edit. Because a value stays in force until the next entry begins, until 7 March PolicyEngine-US used the previous year's allowances for October 2025 and the months after. A benefit calculated for those months before 7 March rested on outdated figures, and nothing in the result showed it.

3.4 What the 72 commits did

To see why the past was changed, we read the message and the changes of each of the 72 commits and gave each one a label for the main thing it did, and a second label where it did two things. Table 3 counts the main labels. One person did the reading, and no second reader checked the labels (Section 6).

Table 3. The 72 commits that changed a period already in effect

Main thing the commit did	Commits	Changes to past entries
A list with one start date was edited	29	32
A wrong value was fixed	23	287
A wrong start date was fixed	10	51
A first date was moved earlier to cover more years	5	143
A forecast was replaced by newer figures	2	4
A file's layout was repaired	2	23
Correct entries were changed as a side effect	1	4

Three groups stand out.

- **Lists with one start date.** Some rules are stored not as dated values but as a list with a single start date. The most common is the list of payments that count as unearned income, dated 2009, which 26 commits changed. In all, 29 commits edited such a list, and 19 of them added a payment PolicyEngine-US had just begun to model, such as a state's supplement to federal disability payments. Because the list has one start date, each addition counts the new payment as income in every year back to 2009. This is how PolicyEngine-US is built, not an error, but it means the list has no history: PolicyEngine-US cannot say what counted in an earlier year if that ever differed.
- **First dates moved earlier.** Five commits moved a series' first date back so that it covered more years. This assumes the value was the same in those earlier years; if it was not, those years are now wrong.
- **Fixes.** Thirty-three commits fixed a wrong value or a wrong start date, and ten of them also did something else, such as fixing both. Section 3.5 looks at what they fixed.

Only one commit changed correct entries by accident: the Maryland commit described at the start of Section 1. A second, labeled a fix, replaced 14 correct utility allowances with wrong values from the official federal table (Section 3.6). We did not check the other corrections against the law (Section 6.2), so these are the two cases we know of.

3.5 The fixes mostly repair the same year's work

Most fixes repaired recent work. Of the 33 fix commits, 29 changed entries that an earlier commit in 2026 had written. Counted by entry, 263 of the 562 corrections changed something entered earlier the same year; the other 299 changed entries older than 2026. Half of the 263 came within 7 days of the entry they changed, so many mistakes were caught quickly. But 44 came more than 90 days later: an entry could stand wrong for months before anyone noticed.

The fixes also reach far back in time. The median fix commit changed a period that had begun 4.5 years before the commit, and 14 of the 33 reached back more than five years. Some series needed more than one pass: 61 were corrected in two or more commits, and Maryland's standard utility allowance in four. Utility allowances alone account for 298 of the 544 corrections to periods already in effect.

Why were the fixes made? Table 4 gives the reason each fix commit states in its message. Just over half, 18 of 33, cite a check against a named source document. Eight give no reason at all.

Table 4. What prompted the fixes, by the commit message

Reason given	Fix commits
A check against a named source document	18
A reviewer or review tool	5
A linked issue	1
Two files disagreed	1
No reason given	8

3.6 Four histories

Counts hide how a single value travels through a year. Four cases show it.

Maryland's utility allowance. On 6 March a commit changed four correct entries, and on 9 March another commit restored them. Maryland's own memo agrees with the restored figures: the allowance rose from $505 to $551 on 1 January 2024 (Maryland Department of Human Services, 2024). That was not the end. On 7 March, the amount for fiscal year 2026 had been entered with a start date of 1 October 2025. Maryland's amounts start on 1 January, and the date was moved to 1 January 2026 only on 24 August. For five and a half months, PolicyEngine-US applied the new amount three months early.

The fiscal year 2026 allowances. These values went through four versions. On 7 March they were entered from a secondary website. On 10 March, 60 of them were fixed, because two columns had been swapped for ten states and regions. On 19 August they were replaced with the official federal table. On 24 and 25 August, 14 of those replacements were reversed after review: according to the commit messages, the federal table listed Wisconsin's later mid-year amounts, shifted New York's regions and misaligned Maine's row. The official source, taken as final, was itself wrong for those entries.

The start of the 2025 work rules. The 2025 law's changes to the work requirement needed a start date, and the federal date was entered three times in February: first as 1 January 2027, then 1 July 2025, then 4 July 2025, the day the law was signed. Alaska has its own entry, because the law gives Alaska and Hawaii a separate arrangement. That entry began as 1 November 2025, became 1 November 2026 on 7 August, went back the same day after review, and became 4 July 2025 on 22 October.

Child support. States choose how child support payments are treated when the benefit is calculated, and PolicyEngine-US keeps a table of each state's choice. A change made in 2023 relabeled the table, from "deducted from net income" to "deducted from gross income", without flipping the values in it. By the fixing commit's own count, 46 of 53 jurisdictions carried the wrong treatment in 2026, until the fix on 24 September. The same commit updated tests that had been written to expect the wrong values. This error is older than the year we studied. It shows how long a wrong entry can stand when nothing prompts a check, and that tests can stand with it.

3.7 Changes that did not hold

Six commits made changes that a later commit reversed exactly: 23 changes in all. Fourteen are the fiscal year 2026 replacements above, and four are the Maryland entries. When one edit is undone exactly by another, one of the two was wrong. In Maryland it was the first. In the fiscal year 2026 case it was the second, the one that was meant to be the correction.

3.8 What the co-author line can and cannot say

Of the 72 commits that changed a period already in effect, 63 carry the Claude co-author line. That is 88 percent, the same share as among all 101 commits that changed a dated value (89 of 101). So the commits that fix are AI-assisted, and so are most of the commits they fix. Because the line does not show which choices the AI tool made and which the people made (Section 1), these counts describe the upkeep of an AI-assisted codebase. They do not measure the AI tool.

4. Do agents damage the past?

The commit record shows two updates in the year that overwrote correct entries. It cannot show how often an AI coding agent, given a dated change to make, would do this on its own. For that we ran a controlled test.

4.1 The design

We chose ten real changes made to PolicyEngine-US in 2026 (Table 5). For each, we built a copy of PolicyEngine-US with that change taken out: its dated entries deleted, its commit reversed or its old text restored. We also removed the 67 test files that would have given the answer away.

The agent then received the copy and a short notice that we wrote, describing the change in plain words the way an agency circular would. Here is the notice for the Maryland correction, in full:

> Maryland does not update its SNAP standard utility allowance (SUA) and limited utility allowance (LUA) at the start of the federal fiscal year. Its new amounts take effect each 1 January. USDA's annual utility allowance tables footnote Maryland as effective 1 January, including for fiscal year 2026, whose Maryland amounts took effect on 1 January 2026.
>
> The model starts each of Maryland's SUA and LUA amounts since 2020 on the previous 1 October, as if Maryland followed the federal fiscal year. The amounts themselves are right; their start dates are wrong.
>
> Fix the dates. Other states are not affected.

Each task was run under two conditions. In condition A, the agent had the notice alone. In condition B, it also had one test it could see and run: one household, checked in a month before the change and a month after it. That made 20 runs, one for each task and condition.

The agent was Claude Code, a coding tool that works through a task on its own without a person approving each step, using the AI model Claude Sonnet 5.5. It could read and edit files in its copy, run the repository's tests, and use two version-control commands that show its own changes (git diff and git status). The copy was built from the files alone, without the repository's history, so the original change could not be looked up. It had no web access.

Table 5. The ten tasks

Task	Change put back	Kind
P1	Energy-assistance route to the largest utility allowance limited to elderly or disabled households	Logic
P2	Arizona's gross income limit raised to 200% of the poverty line	Value
P3	Noncitizen eligibility narrowed by the 2025 law	List of statuses
P4	Fiscal year 2027 cost-of-living figures	Value, six files
P5	Wisconsin's May 2026 utility allowances	Value, eight files
Q1	Former foster youth exemption from the work requirement	Logic
Q2	California applies the 2025 work-requirement changes from June 2026	Logic, one state
Q3	California applies the 2025 noncitizen changes from April 2026	Logic, one state
Q4	California's broad eligibility route has no net income test	Correction, back to 2015
Q5	Maryland's utility allowances start each 1 January	Correction, back to 2020

Eight tasks add a rule from some date onward. The last two, Q4 and Q5, are corrections: the notice says the record is wrong for past years, and the agent must change the past on purpose and stop in the right place.

4.2 Three ways of grading

Each run was graded in three ways.

- **Household check.** For each task we wrote five to seven example households and ran them through the agent's copy and through the maintainers' version, for six to eleven months on both sides of the change's date. One check is one household, one month and one result. Months before the date test whether the past was kept; months from the date on test the new rule.
- **History check.** Every dated entry earlier than the change's date was compared before and after the run.
- **Maintainers' tests.** We ran the test files we had removed, the tests the maintainers wrote for the same change, on the agent's copy: 957 test cases across the ten tasks. They fail on the copy with the change removed and pass on the maintainers' version.

4.3 The past was left alone

Table 6. Results of the 20 runs. A is the notice alone; B adds the visible test. "Differ" counts checks whose result differs from the maintainers' version.

Task	Before the date, differ (A / B)	From the date, differ (A / B)	Earlier entries changed	Maintainers' test cases	Real failures (A / B)
P1	0 of 54 / 0 of 54	0 of 54 / 0 of 54	0	146	4 / 0
P2	0 of 45 / 0 of 45	0 of 45 / 0 of 45	0	18	0 / 0
P3	0 of 36 / 0 of 36	0 of 36 / 0 of 36	0	148	0 / 0
P4	0 of 105 / 0 of 105	0 of 105 / 0 of 105	0	208	0 / 0
P5	0 of 45 / 0 of 45	0 of 45 / 0 of 45	0	29	0 / 0
Q1	not scored	24 of 60 / 0 of 60	0	77	3 / 1
Q2	0 of 30 / 0 of 30	0 of 120 / 0 of 120	0	83	0 / 0
Q3	0 of 20 / 0 of 20	0 of 60 / 0 of 60	0	107	0 / 0
Q4	none to score	0 of 168 / 0 of 168	0	112	0 / 0
Q5	0 of 15 / 0 of 15	0 of 150 / 0 of 150	0	29	0 / 0

"Real failures" leaves out test cases that fail only because they name a helper variable, an in-between quantity from the maintainers' own design (1 in P1, 23 in Q3 and 16 in Q4, in each condition), or only because of the July 2025 date convention (1 in P3). Section 4.5 explains both.

Across the 20 runs there were 700 checks in months before the change, and none differed from the maintainers' version. No run changed or removed a dated entry earlier than its change. In the eight tasks that add a rule, the agent nearly always worked by addition: it placed a new dated entry beside the old ones and left those alone.

The two corrections are the harder test, because there the agent has to edit history and stop in the right place. In all four correction runs, every check matched the maintainers' version, and no dated entry earlier than the corrected span was changed.

From the change's date onward there were 1,686 checks. Twenty-four differed, all in one run (Q1, condition A), for a reason explained next.

4.4 Three errors, all about fit

The agent made three errors. None concerns a date. Each concerns fit: how the new rule sits with the code that was already there.

A rule built in the wrong place (P1, condition A). The 2025 law limited one route to the largest utility allowance to households with an elderly or disabled member. The maintainers changed the step that chooses a household's allowance type. The agent instead turned a switch that applied to whole states into a condition on each household. Every household in our check got the right answer. But four of the maintainers' tests failed: households in states the switch did not cover kept the full allowance, and two households lost a state rule they should have kept. With the visible test, in condition B, the agent chose the maintainers' design and passed.

An exemption added to one list and missed in a second (Q1, both conditions). The formula for the work-requirement exemption holds two lists. One has the exemptions that applied before the 2025 law. The other has those that Alaska and Hawaii may keep for a time after adopting it. Former foster youth belong in both.

In both runs, the agent added them to the first list only, so a 22-year-old former foster youth in Anchorage in January 2026 is not exempt when they should be. Our households lived in Texas and Florida and never reached that part of the code. The second list sits a few lines below the line the agent edited, and it already names the other two exemptions that the 2023 act added.

A new input where one existed (Q1, condition A). An input is a fact about a household that the user supplies, such as a person's age. PolicyEngine-US already had an input recording that a person was in foster care. The agent judged that it did not say whether the person was in care at age 18, created a second input and wrote the rule against that. The rule is right when the new input is filled in. But existing household data and the maintainers' tests fill in the old input, so for them the exemption never switches on. This is the run with the 24 differing checks; with the new input linked to the old one, all 60 checks match. In this run the operating system blocked the test runner, so the agent could not run any tests. With the visible test, which uses the existing input, the agent used it too.

4.5 What each check could see

The household check caught one of the three errors. The other two gave the right answer for every household we had written, because our households did not include the cases where the code went wrong.

The maintainers' tests caught all three. They cover states, household types and parts of the code that a check written for one change does not. They also failed for one reason that is not an error. In tasks Q3 and Q4, 39 test cases per condition failed only because they name helper variables: in-between quantities from the maintainers' own design, which an agent solving the task another way has no reason to create. With those names removed from the expected results, every remaining check in those tests passed. Eight test cases check nothing but a helper variable and could not be re-run this way. So an existing test suite is a strong check of fit, but it marks a different valid design as wrong unless it is adjusted like this.

The visible test caught none of the errors directly. Its effect came earlier: in both tasks where the agent without it built the rule differently from the maintainers (P1 and Q1), the agent with it took the maintainers' route.

One difference came from the date convention of Section 2. In task P3, the agent dated the change 4 July 2025, the day the law was signed, in both runs. The maintainers used 1 July. The two versions differ for July 2025 only, and we did not score that month.

4.6 A disagreement about the past

In one task the agent and the maintainers disagreed about the months before the change, and we did not score those months. The former foster youth exemption began on 1 September 2023, and in both runs the agent dated it from then. The maintainers' version exempts former foster youth in earlier months as well. On this point the agent's code follows the 2023 act and the maintainers' code does not. Section 5.1 explains why.

5. What goes wrong instead

If an agent adding one dated change leaves the past alone, where do wrong answers about past months come from? In PolicyEngine-US we found four sources. The first is new in this section; the other three gather what Sections 3 and 4 showed.

5.1 A formula that sets the dates aside

Section 2 described a formula that names a date and reads the rules as they stood on that day. We searched all 115 SNAP formula files for such reads. Together they read the rules 62 times, and four of those reads use a date other than the month being calculated:

- **Income standards.** One formula reads the poverty guidelines as of 1 October of the fiscal year. The date is worked out from the month being calculated and follows SNAP's practice of adopting new guidelines each October. It is correct.
- **Waived areas.** One formula reads a list of areas excused from the work requirement as of 3 July 2025, and only for a setting that asks what would have happened without the 2025 law. It does not affect results under the law as it is.
- **Children in the household.** One formula reads the age limit for a child as of 1 June 2025. The limit was 18 from 1997 until the 2025 law, so the answer is right for every earlier month. It would become wrong only if an earlier change were ever recorded.
- **The work-requirement exemption.** One formula reads the exemption age and the foster-care age as of 1 June 2025. This one gives wrong answers.

The rule's history. Adults without dependents can receive SNAP for only three months in any three years unless they meet a work requirement, a limit known as the time limit. Older adults are exempt. From 1997 adults over 50 were exempt. The statute says "over 50" and the regulation "50 years of age or older", so the two disagree about a person aged exactly 50 (7 U.S.C. § 2015(o)(3)(A), 2017; 7 C.F.R. § 273.24(c)(1), 2017). The Fiscal Responsibility Act of 2023 raised that age in steps: 51 from 1 September 2023, 53 from 1 October 2023 and 55 from 1 October 2024 (Fiscal Responsibility Act of 2023, 2023, § 311; U.S. Department of Agriculture, Food and Nutrition Service, 2024). The same act also exempted homeless people, veterans and young adults who had been in foster care, from 1 September 2023. PolicyEngine-US's dated record holds the age steps correctly.

What the formula does. For any month in which the 2025 law does not yet apply, the formula uses the values of 1 June 2025. The fixed date has a purpose. California applied the 2025 law only from June 2026, so for a California month in early 2026 the rules of that month are the new ones, while the formula needs the old ones; reading the rules as of June 2025 supplies them. But it supplies the same June 2025 values for 2022, when they were not the rules. So the exemption age is 55 in every earlier month, and the three added exemptions apply in every earlier month.

What it changes. We ran 11 single adults in Texas through 12 months between March 2019 and June 2025 and compared the formula's answer with the dated record for each month (Table 7). One person in one month is a case-month.

Table 7. The exemption formula against the dated record

Group	Case-months	Formula differs from the record	Of these, while the time limit was suspended
Adults aged 50 to 54	60	35	15
Homeless, veteran, former foster youth	36	15	9
Others (aged 30, 49, 55)	36	0	0
Total	132	50	24

The formula and the record disagree in 50 of 132 case-months, and the two groups in which they disagree need to be read differently.

For adults aged 50 to 54, the error is plain. In 35 case-months the record exempts them and the formula does not, and PolicyEngine-US computes no benefit for them. Five of the 35 concern a 50-year-old, whom the statute and the regulation treat differently (the dated record follows the regulation); the other 30, at ages 51 to 54, are wrong under either reading. An exempt adult in the same months would receive between $192 and $345 a month. One complication does not change this. During the pandemic, Congress suspended the time limit from 1 April 2020 to 30 June 2023 (Families First Coronavirus Response Act, 2020, § 2301), and PolicyEngine-US does not represent the suspension. But these adults were exempt by age with or without it, so they qualified in all 35 case-months.

For the three added exemptions, the label is wrong but the benefit mostly is not. In 15 case-months before September 2023, the formula exempts a person the law had not yet exempted. Nine of those months fall inside the suspension, when nobody was being cut off. Three more are in August 2023, when the limit had just restarted and the first cut-off was still two months away. In those twelve, the amount PolicyEngine-US computes matches what the law provided. Only the three cases from March 2019 show a benefit that the time limit could have stopped.

The tests agree with the formula. The maintainers have 18 tests of this rule for 2022 and 2024. None checks an adult aged 50 to 53 before the exemption age rose. Three state the formula's answer: a homeless person and a veteran are expected to be exempt in 2022, before the law exempted them, and a 54-year-old in Texas is expected not to be exempt in 2024, a year in which the record exempts that age until October.

How it arrived. The fixed-date read and the correct record arrived on the same day, 23 February 2026, in two commits from one batch of work. The first carries the Claude co-author line. The second, whose whole message is the word "fix", does not. The first commit replaced a set of California-specific entries with the read at 1 June 2025. At that moment the record said 55 from 2019, so the read and the record agreed for every earlier month. The second commit corrected the record to the stepped history and left the read in place. From then on the record was right, and no result changed. The steps of 50, 51 and 53 are in the record, and no formula reads them. In August the read moved, unchanged, into the formula's current file. The work was about the 2025 law and the states adopting it late, and no test asked about earlier years.

This is the Maryland case turned around. There, an edit changed a record that was right. Here, the record was put right and the answer stayed wrong, because the correction was made in the entries while the answer comes from the formula.

It also explains the disagreement of Section 4.6. Asked to add the foster youth exemption, the agent gave it a start date of 1 September 2023 in both runs. The maintainers' formula gives it none.

Already reported. The fixed-date read had been reported in the project's issue tracker on 28 September 2026, six weeks before the version we studied (PolicyEngine, 2026b), and a correction was proposed on 13 November 2026, the day of that version (PolicyEngine, 2026c). We found the error without knowing of the report. When we wrote, the correction had not been merged, and the formula on the project's main branch still read the rules as of 1 June 2025 (checked on 20 November 2026). The proposed correction follows the statute on age 50, so it would also settle the five disputed case-months the other way.

5.2 Rules with no date

The second source is a rule written with no date at all. The three added exemptions are one example: inside the formula they are simply conditions, and nothing ties them to September 2023.

The lists of Section 3.4 are another. Twenty-nine commits edited a list with a single start date, most often the list of unearned income sources, dated 2009. Whatever such a list says today, it says for every year since 2009.

A search for fixed dates cannot find these, because there is no date to find. We saw the undated exemptions only because they sit in the same formula as a fixed-date read.

5.3 Entries that arrive late or settle slowly

The third source is the record itself, in two ways that involve no damage to correct entries.

- **Late entry.** New periods were entered a median of 157 days after they took effect. Until an entry exists, PolicyEngine-US answers with the value before it.
- **Correction in passes.** The fiscal year 2026 allowances took four passes to settle, and one pass replaced right values with wrong ones from an official table. Forty-four corrections came more than 90 days after the entry they fixed.

The year's accidental damage belongs here too. It came from a commit that added values for every state at once, on a schedule that fits most states but not Maryland.

5.4 Rules that do not fit the code around them

The fourth source is the one the agent showed in Section 4: a rule placed in the wrong part of the code, a rule added to one list and missed in a second, and an input created beside one that already existed. None of these is about time. Each produced a wrong answer for some household in some month after the change.

5.5 What found each one

Table 8. Sources of wrong answers, and what found them in this study

Source	Example	What found it	Did the maintainers' tests find it?
A formula reads the record at a fixed date	Exemption formula	Comparing the formula with the dated record for earlier months	No. Three tests expect the formula's answer.
A rule has no date	Early exemptions; single-date lists	Reading the statute for the start date	No.
An entry arrives late	Fiscal year 2026 allowances	Comparing each entry's date with the date it was entered	Not examined.
A bulk edit reaches correct entries	Maryland	Comparing entries already in effect before and after the commit	The commit was merged. Tests were added with the repair.
A rule does not fit the surrounding code	The three agent errors	The maintainers' full test suite	Yes, all three.

Two points follow from Table 8. First, the checks that found the date errors are simple comparisons that a script can run: entries in effect before and after a commit, a formula's answer for an earlier month against the dated record, and an entry's date against the day it was entered. None needs a judgment about the law, except the question of when a rule began.

Second, the tests split. For errors of fit, they were the only check that worked. For errors about the past, they were silent or wrong. Twice in this year, in the exemption formula and in the child support table, tests stated an answer that the law does not support, and passed.

5.6 What the three checks would have flagged

Section 8 proposes three checks that a script can run on every change. To see whether they would help a maintainer or bury one in alerts, we ran each over the year (Table 9). This is a look back with the errors already known: it shows what the checks would have flagged, not how a reviewer would have responded.

Table 9. What each check would have flagged in 2026

Check	What it flags	Flagged in 2026	Real problems among the flags
A. Compare the entries already in effect before and after each commit	Commits that change a period already in effect	72 of 101 commits	2 damaged correct entries; 33 fixed an earlier error; 4 more made changes later reversed exactly
B. Compare each formula's answers for earlier months with the dated record	Formulas that read the rules at a named date	4 of 115 formula files	1 gives wrong answers, in 50 of 132 case-months
C. Compare each entry's date with the day it was entered	New periods entered after they took effect	616 of 736 new periods, in 12 commits	All 616: each left a stretch of outdated answers, a median of 157 days

Each check flagged the errors of its kind that this study confirmed. Check A flagged both commits that damaged correct entries, the Maryland change and the 19 August replacement, on the day they were made, days before either was repaired. Check B singled out the exemption formula among 115; for the other three it flagged, we judged the effect by reading the code and the rule's history (Section 5.1), not by running them. Check C flagged every late entry, including the 443 allowances entered on 7 March.

The checks differ in what they ask of a reviewer. Check B is nearly free: four flags, one of them real. Check C's flags are all real, because a late entry gives outdated answers by definition; what it adds is how long. Check A is the noisy one. Most of its 72 flags were legitimate work, and only 2 were damage. But the 33 fixes it flagged are themselves changes to the past, which the year showed can go wrong, and the flags came in 23 weeks, about three commits in a week that had any. Reading three commits a week that touch the past is a modest price for catching damage that the tests let through.

None of the three would have found the child support table, the rules with no date, or the agent's errors of fit (Table 8). Those need the law, a reading of the code, or the maintainers' tests.

6. Limits

The study is small in every direction. This section sets out what each part can and cannot show.

6.1 Scope

- **One codebase, one program, one year.** Everything here comes from the SNAP part of PolicyEngine-US between 1 January and 13 November 2026. Another policy model, another program or another year could look different.
- **A calculator, not a benefit system.** PolicyEngine-US is used for analysis and for estimating benefits; it does not decide anyone's case. The same questions could be asked of the systems that do, but this study says nothing about them.
- **No comparison year.** We did not study a year of upkeep without AI assistance, so we cannot say whether the rates in Section 3 are higher or lower than they would otherwise be.

6.2 The commit study

- **One reader.** The labels in Table 3 and the reasons in Table 4 come from one person reading each commit. No second reader checked them.
- **Messages, not causes.** The reason for a fix is what its commit message says. Eight fix commits give none.
- **Right and wrong were not judged.** A commit counts as a fix because it presents itself as one. We did not check the final values against the law, except where a later commit reversed them and in the cases checked against sources in Section 5.
- **The sorting has known weak points.** A moved date is matched to its new date by its value, and 11 of the 190 moves had more than one possible match. A renamed series looks like one series removed and another added; 14 such renames changed or dropped 7 entries that are not counted as corrections. The 99 changes whose value is a list were not checked the second way (Section 3.1).
- **SNAP folders only.** We counted parameter files in PolicyEngine-US's SNAP folders. A few SNAP values are stored elsewhere, such as the income limits for broad-based categorical eligibility (a route into SNAP through other assistance), and changes to them are not counted.
- **Commits are uneven units.** One piece of work can arrive as one commit or as several, as on 23 February. Counting commits treats them alike.

- **The co-author line.** It shows that an AI tool took part. Over the year it names several Claude models. It does not show who chose a value, who checked it, or how the tool was used.

6.3 The controlled test

- **Twenty runs, one AI model, one run each.** With one run for each task and condition, we cannot measure how much results vary from run to run. The finding is that no damage to the past was seen in 20 runs. It is not a rate.
- **A different setting from the commits.** The runs used Claude Sonnet 5.5 working alone from a notice. The year's commits were made by people working with other Claude models. The two are not a like-for-like comparison.
- **Our notices.** We wrote them. They are clear and complete, and they state the date. Real notices are often none of these.
- **Our households.** We chose them, and they missed the Alaska case.
- **The yardstick is the maintainers' version.** It is itself wrong in places, as Section 5 shows, which is why one task's earlier months were not scored.
- **Hints.** In two tasks, a few code comments naming the change remained in the copy given to the agent.
- **Prior exposure.** The ten changes are public, and Claude Sonnet 5.5's training data runs to June 2026 (Anthropic, 2026). Four of the ten (P3, Q2, Q3 and the Maryland dates of Q5) were in the repository before then, so the AI model may have seen them during training. The other six were committed afterwards. All three errors came in tasks committed afterwards (P1 and Q1). The numbers are too small to say whether that matters.
- **One run without tests.** In one run, the operating system blocked the test runner.

6.4 The formula audit

- **Yardstick.** We compared the formula with PolicyEngine-US's own dated record. We checked the age steps and the September 2023 start against the statute and USDA's final rule, and no other values.
- **What PolicyEngine-US leaves out.** It does not represent the pandemic suspension of the time limit (reported in the project's issue tracker on 18 November 2026 (PolicyEngine, 2026d)), and it treats a person who fails the work requirement as ineligible at once, where the law allows three months.
- **Coverage.** The test used single adults in one state, Texas, in 12 months.
- **The search finds named dates only.** Rules written with no date are not found by it, and we do not know how many there are.

6.5 How to read the results

The counts describe one repository in one year. They show that revising the past was routine there, that two updates in the year damaged correct entries, that 20 runs of an AI coding agent did not, and that one formula bypassed the record. They are observations. Each would need more codebases, more AI models and repeated runs before it could be stated as a rate.

7. Related work

AI tools that write legal software. The nearest work asks language models (AI models trained on large amounts of text) to write or update tax software. Gogani-Khiabani and colleagues had models produce the 2021 version of tax functions from official publications, with or without the 2020 code to start from, and scored the 2021 result (Gogani-Khiabani et al., 2024). Each tax year is a separate program there, so the question of whether the earlier year stays correct does not arise. A later paper from the same group translates tax law into code with several cooperating agents (Gogani-Khiabani et al., 2026). Others have used language models to extract SNAP and Medicaid rules and generate code from them (Kennan et al., 2025; Ratna et al., 2024). All of these start from the law and build new code. We start from code that already holds years of dated rules and ask what happens to those years.

Time in legal question answering. A recent group of studies shows that language models answering legal questions often apply the wrong version of a rule, one that is out of date or not yet in force (Cymbler et al., 2026; C. Li et al., 2026; Prior et al., 2026; Sobhani et al., 2026). Related work updates a language model's legal knowledge after a law is amended (C. Wang et al., 2024). These studies test an answer given once to one question. In a rules codebase, the choice of version is written into a program that then answers for everyone, so an error of the same kind lasts until someone finds it.

Code generation across versions. Benchmarks (standard sets of tasks used to compare tools) such as GitChameleon, VersiCode and CodeUpdateArena test whether a language model writes code for the right version of a software library (Liu et al., 2024; Misra et al., 2025; Wu et al., 2024). The task there is to target one version. In benefit rules, the old version and the new one must both stay available, each for its own months.

Agent-written code in real projects. Large studies now describe what AI coding agents contribute on GitHub, the site where many open-source projects keep their code. Watanabe and colleagues examined 567 pull requests (proposed changes submitted for review) written with Claude Code in 157 projects (Watanabe et al., 2025), and the AIDev dataset collects more than 450,000 agent pull requests from five tools (H. Li et al., 2025). They measure how often such changes are accepted and revised across many projects. We follow one project for a year and look at one kind of content, the dated values. Closer to our controlled test, Wang, Pradel and Liu found that 7.8 percent of patches counted as solving an issue in a widely used benchmark fail the developers' own tests, because the benchmark runs only the test files changed with the fix (Y. Wang et al., 2026). Two of our three agent errors are the same thing in a different field: each gave the right answer on the check written for the change and failed the maintainers' tests.

Two kinds of time. Database research has long separated valid time, when a fact holds in the world, from transaction time, when it was recorded (Snodgrass & Ahn, 1985). PolicyEngine-US's parameter files store valid time: the date each value took effect. The repository's history supplies transaction time: the day each value was entered. Our measures of late entry and of correction are measures of the gap between the two. For legal code in particular, Monat, Fromherz and Merigoux formalized date arithmetic and found ambiguous date calculations in an implementation of French housing benefits (Monat et al., 2024). Their concern is how a date is calculated. Ours is which dated version of a rule is read.

Testing rules as code. Tax software has been tested by checking that related inputs give consistently related outputs, without needing a known right answer (Tizpaz-Niari et al., 2023). Thapa (2026a) compared open-source Social Security benefit calculators, PolicyEngine-US among them, with the agency's official

calculator, and traced the other calculators' disagreements to rounding order, date handling, and limits that were missing or derived instead of taken from the published tables.

Practice. The repository we studied carries written instructions for AI coding agents and loads a shared set of skills (packaged instructions) for encoding policy (PolicyEngine, 2026a). We found no published evaluation of them. We also found no study, of people or of AI tools, that measures what upkeep does to the earlier periods of a dated rules codebase.

8. Conclusion

We asked whether the AI-assisted upkeep of a policy model damages its record of past rules. For one policy model, one program and one year, the answer has three parts.

The past is revised all the time. Seventy-two commits changed periods already in effect, most often to repair entries made earlier the same year, and new periods arrived a median of 157 days after they took effect.

Outright damage was rare. Two commits in the year altered correct entries, one by accident and one by trusting a faulty official table, and both were reversed within a week. An AI coding agent asked to add a dated change did not damage the past in any of 20 runs, including four runs that required correcting it.

The wrong answers came from elsewhere. A formula read the rules at a fixed date, so a correction made to the record never reached a result. Rules were written with no date. Entries arrived late or took several passes to settle. And new rules did not always fit the code around them, which the maintainers' own tests revealed and a check written for the change mostly did not.

The study leaves three checks that a script can run on any policy model built this way: compare the entries already in effect before and after each change; compare each formula's answers for earlier months with the dated record; and compare each entry's date with the day it was entered. Run over the year, they would have flagged both commits that damaged correct entries, the formula that bypassed the record and every late entry (Section 5.6). It also leaves one caution about tests. A test protects the answer it states, and twice in this year a test stated an answer taken from PolicyEngine-US itself and not from the law.

Whether these patterns hold for other policy models, other programs and other AI tools is the question that follows.

Data availability

The commit data, the label for each of the 72 commits, the audit cases, the pilot tasks with each run's changes and grades, and all scripts are released at https://doi.org/10.5281/zenodo.24406575 (Thapa, 2026b).

Declarations

Competing interests. The author declares that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

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

AI-based tools assisted with the programming and the writing of this paper.

References

- 7 C.F.R. § 273.15 (2024). Fair hearings. https://www.govinfo.gov/content/pkg/CFR-2024-title7-vol4/xml/CFR-2024-title7-vol4-sec273-15.xml
- 7 C.F.R. § 273.17 (2024). Restoration of lost benefits. https://www.govinfo.gov/content/pkg/CFR-2024-title7-vol4/xml/CFR-2024-title7-vol4-sec273-17.xml
- 7 C.F.R. § 273.18 (2024). Claims against households. https://www.govinfo.gov/content/pkg/CFR-2024-title7-vol4/xml/CFR-2024-title7-vol4-sec273-18.xml
- 7 C.F.R. § 273.24(c)(1) (2017). Time limit for able-bodied adults: exception for persons under 18 or 50 years of age or older. https://www.govinfo.gov/content/pkg/CFR-2017-title7-vol4/xml/CFR-2017-title7-vol4-sec273-24.xml
- 7 C.F.R. pt. 275, subpt. C (2024). Quality control (QC) reviews. https://www.govinfo.gov/content/pkg/CFR-2024-title7-vol4/xml/CFR-2024-title7-vol4-sec275-10.xml
- 7 U.S.C. § 2015(o)(3)(A) (2017). Work requirement: exception for persons under 18 or over 50 years of age. https://www.govinfo.gov/content/pkg/USCODE-2017-title7/html/USCODE-2017-title7-chap51-sec2015.htm
- Anthropic. (2026). Claude Sonnet 5.5: Model overview. Retrieved from https://platform.claude.com/docs/en/models/sonnet-5-5/overview. Accessed November 15, 2026.
- Ark. Code Ann. § 26-51-502(b)(1). Household and dependent care expenses. Retrieved from https://codes.findlaw.com/ar/title-26-taxation/ar-code-sect-26-51-502/. Accessed November 20, 2026.
- Cymbler, R., Guez, D., & Fabre, L. (2026). Temporal misgrounding in legal RAG: A versioned-corpus benchmark for French tax law. ICML 2026 Workshop on AI for Law. arXiv preprint. https://doi.org/10.48550/arXiv.2608.09393
- Families First Coronavirus Response Act, Pub. L. No. 116-127, § 2301 (2020). https://www.govinfo.gov/content/pkg/PLAW-116publ127/html/PLAW-116publ127.htm
- Fiscal Responsibility Act of 2023, Pub. L. No. 118-5, § 311 (2023). https://www.govinfo.gov/content/pkg/PLAW-118publ5/html/PLAW-118publ5.htm
- Gogani-Khiabani, S., Dewangan, V., Olson, N., Trivedi, A., & Tizpaz-Niari, S. (2024). Technical challenges in maintaining tax prep software with large language models. 14th Annual IRS/TPC Joint Research Conference on Tax Administration. arXiv preprint. https://doi.org/10.48550/arXiv.2504.18693
- Gogani-Khiabani, S., Trivedi, A., Saha, D., & Tizpaz-Niari, S. (2026). An LLM agentic approach for legal-critical software: A case study for tax prep software. 48th International Conference on Software Engineering (ICSE 2026). arXiv preprint. https://doi.org/10.48550/arXiv.2509.13471
- Kennan, A., Singh, L., Garcia Guevara, A., Ahmed, M., & Goodman, J. (2025). AI-powered rules as code: Experiments with public benefits policy. Beeck Center for Social Impact + Innovation and Massive

Data Institute, Georgetown University. Retrieved from https://digitalgovernmenthub.org/publications/ai-powered-rules-as-code-experiments-with-public-benefits-policy/. Accessed November 15, 2026.

- Li, C., Feng, Z., Huang, Y., Ye, Y., & Xie, H. (2026). LexKairos: Benchmarking legal temporal capabilities in LLMs. arXiv preprint. https://doi.org/10.48550/arXiv.2608.09106
- Li, H., Zhang, H., & Hassan, A. E. (2025). The rise of AI teammates in software engineering (SE) 3.0: How autonomous coding agents are reshaping software engineering. arXiv preprint. https://doi.org/10.48550/arXiv.2507.15003
- Liu, Z. L., Pandit, S., Ye, X., Choi, E., & Durrett, G. (2024). CodeUpdateArena: Benchmarking knowledge editing on API updates. arXiv preprint. https://doi.org/10.48550/arXiv.2407.06249
- Maryland Department of Human Services. (2024). 2024 increase in the Supplemental Nutrition Assistance Program (SNAP) and long-term care medical assistance, waiver medical assistance (MA) utility allowances (Family Investment Administration Information Memo No. 24-32, issued February 15, 2024, effective January 1, 2024). https://dhs.maryland.gov/documents/FIA/Action%20Transmittals-AT%20-%20Information%20Memo-IM/AT-IM2024/24-32%20IM%20SUA%20LUA%20INCREASE.pdf
- Misra, D., Islah, N., May, V., Rauby, B., Wang, Z., Gehring, J., Orvieto, A., Chaudhary, M., Muller, E. B., Rish, I., Kahou, S. E., & Caccia, M. (2025). GitChameleon 2.0: Evaluating AI code generation against Python library version incompatibilities. arXiv preprint. https://doi.org/10.48550/arXiv.2507.12367
- Monat, R., Fromherz, A., & Merigoux, D. (2024). Formalizing date arithmetic and statically detecting ambiguities for the law. European Symposium on Programming (ESOP 2024). arXiv preprint. https://doi.org/10.48550/arXiv.2403.08935
- Okla. Stat. tit. 68, § 2357.43 (2021). As reproduced in Oklahoma Senate Bill 367, floor version (2026). https://oklegislature.gov/cf_pdf/2025-26%20FLR/SFLR/SB367%20SFLR.PDF
- PolicyEngine. (2026a). PolicyEngine-US (Version 2.118.10, commit 47c4e8720, including its instructions for AI coding agents, CLAUDE.md and AGENTS.md, and the shared skills it loads) [Computer software]. GitHub. https://github.com/PolicyEngine/policyengine-us
- PolicyEngine. (2026b). SNAP ABAWD: pre-HR1 rules read from a fixed 2025-06-01 snapshot apply the FRA's final rules before they took effect (2023-2024) and never let the FRA changes sunset on 2030-10-01 (Issue No. 9662) [GitHub issue]. https://github.com/PolicyEngine/policyengine-us/issues/9662. Accessed November 20, 2026.
- PolicyEngine. (2026c). Correct historical SNAP age and FRA exemptions (Pull Request No. 10858) [GitHub pull request, not merged]. https://github.com/PolicyEngine/policyengine-us/pull/10858. Accessed November 20, 2026.
- PolicyEngine. (2026d). The model applies the SNAP ABAWD time limit from April 2020 through June 2023, when federal law suspended it (Issue No. 11120) [GitHub issue]. https://github.com/PolicyEngine/policyengine-us/issues/11120. Accessed November 20, 2026.
- Prior, M., Schultz, A., & Grabmair, M. (2026). Asking for an old friend: Diagnosing and mitigating temporal failure modes in LLM-based statutory question answering. International Conference on Artificial Intelligence and Law (ICAIL 2026). arXiv preprint. https://doi.org/10.48550/arXiv.2605.23497
- Pub. L. No. 119-21 (2025). https://www.congress.gov/119/plaws/publ21/PLAW-119publ21.pdf

- Ratna, S., Weeks, W. B., Lavista Ferres, J., Chopra, A., & Pereira, M. (2024). A methodology for using large language models to create user-friendly applications for Medicaid redetermination and other social services. International Journal of Public Health, 69, Article 1607317. https://doi.org/10.3389/ijph.2024.1607317
- Snodgrass, R., & Ahn, I. (1985). A taxonomy of time in databases. In Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data (pp. 236–246). https://doi.org/10.1145/318898.318921
- Sobhani, M. E., Sayeedi, M. F. A., Chowdhury, F. H., Arefeen, M. A., Sadeque, F., Bari, M. F., & Shatabda, S. (2026). Time present and time past: Benchmarking large language models on temporally evolving document understanding. arXiv preprint. https://doi.org/10.48550/arXiv.2608.08512
- Thapa, S. J. (2026a). How accurate are open-source Social Security calculators? Zenodo preprint. https://doi.org/10.5281/zenodo.24376858
- Thapa, S. J. (2026b). Rewriting the past: Replication package [Data set]. Zenodo. https://doi.org/10.5281/zenodo.24406575
- Tizpaz-Niari, S., Monjezi, V., Wagner, M., Darian, S., Reed, K., & Trivedi, A. (2023). Metamorphic testing and debugging of tax preparation software. In Proceedings of the 45th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS 2023) (pp. 138–149). https://doi.org/10.1109/ICSE-SEIS58686.2023.00019
- U.S. Department of Agriculture, Food and Nutrition Service. (2024). Supplemental Nutrition Assistance Program: Program purpose and work requirement provisions of the Fiscal Responsibility Act of 2023 [Final rule]. Federal Register, 89, 102342 (December 17, 2024). https://www.govinfo.gov/content/pkg/FR-2024-12-17/pdf/2024-29072.pdf
- Wang, C., Su, W., Hu, Y., Ai, Q., Wu, Y., Luo, C., Liu, Y., Zhang, M., & Ma, S. (2024). LeKUBE: A legal knowledge update benchmark. arXiv preprint. https://doi.org/10.48550/arXiv.2407.14192
- Wang, Y., Pradel, M., & Liu, Z. (2026). Are "solved issues" in SWE-bench really solved correctly? An empirical study. 48th International Conference on Software Engineering (ICSE 2026). arXiv preprint. https://doi.org/10.48550/arXiv.2503.15223
- Watanabe, M., Li, H., Kashiwa, Y., Reid, B., Iida, H., & Hassan, A. E. (2025). On the use of agentic coding: An empirical study of pull requests on GitHub. arXiv preprint. https://doi.org/10.48550/arXiv.2509.14745
- Wu, T., Wu, W., Wang, X., Xu, K., Ma, S., Jiang, B., Yang, P., Xing, Z., Li, Y.-F., & Haffari, G. (2024). VersiCode: Towards version-controllable code generation. arXiv preprint. https://doi.org/10.48550/arXiv.2406.07411