CPAT-AI: Rebuilding a Policy Model as a Transparent, Testable Tool

What two AI-assisted rebuilds of the Climate Policy Assessment Tool found

Rebuilt twice in October 2026 under one rule (fix bugs, copy and log deliberate choices), legacy CPAT turned out to hold 228 bugs, 24 of them serious. Building it a second time found seven bugs in the first rebuild.

Stephen J. Stretton · writing in a personal capacity

Climate Fiscal Policy Brief No. 5.2 · October 2026

CPAT shapes reform advice, but no one could audit its legacy workbook

180+

countries covered: carbon taxes, fuel subsidy reform and related policies

21 MB

legacy workbook; its main sheet runs past row 18,000

~400

pages of methodology, which had drifted from the workbook

Formulas differ from row to row, so checking one row says little about the next. Audits of large spreadsheets almost always find errors.

Source: CPAT Documentation (2023); CPAT-AI task log (2026); Panko (1998); Powell, Baker and Lawson (2008).

One rule kept the rebuild faithful: fix bugs, copy deliberate choices, decide departures in writing

A bug is a formula that does not compute what it means to (a wrong row, a term applied twice). Fixed, never copied.

An issue is a deliberate choice open to question (an old data vintage). Copied exactly and logged for the modelers.

Any other departure needs a dated decision by the model's named owner.

So every difference from the original has a code and a reason a reviewer can look up.

Source: CPAT-AI norms, section 0; working paper 5.2, The replication rule.

Two rebuilds and shared codes turn every comparison into a table join

Excel-AI: a readable workbook written by code; one short formula serves a whole block of cells.

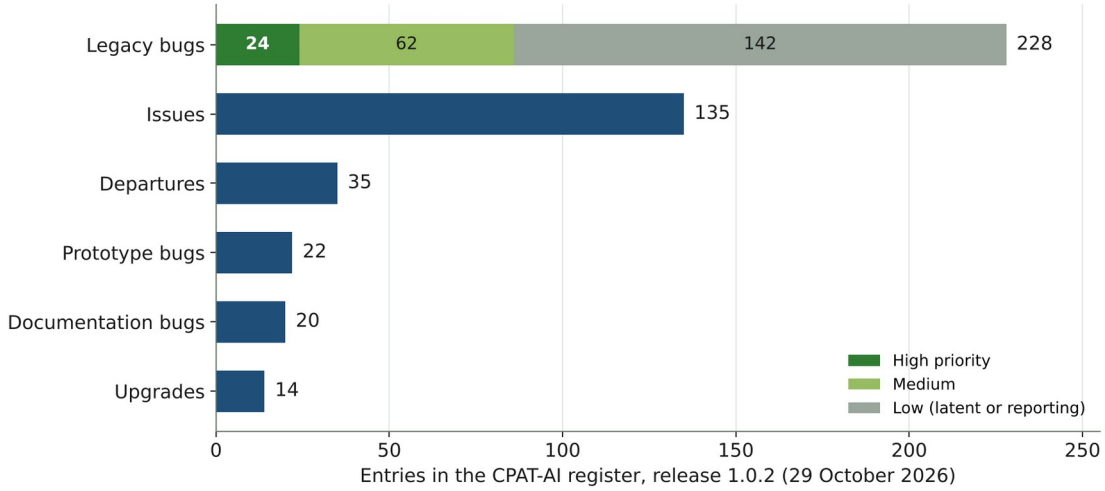
Coded CPAT: a Python model written from Excel-AI's equation list, one function per equation.

One code per input, variable and output, the same in legacy, Excel and Python: e.g. egy.mit.co2.tot.tot.2 is Egypt's total CO2 in scenario 2.

So the three models can be compared after every change, not once at the end.

Source: CPAT-AI repository, release 1.0.2 (29 October 2026); working paper 5.2, Shared codes.

The rebuild registered 454 entries; 228 are bugs in the legacy model, 24 of high priority



CPAT-AI register, release 1.0.2 (29 October 2026).

High (24): wrong results in standard runs.

Low (142): latent: they fire when a ministry switches on an option the default run does not use.

Also 20 errors in the published documentation.

Source: CPAT-AI register (Tracking_Legacy_Bugs_v1.0.2); working paper 5.2, Table 1.

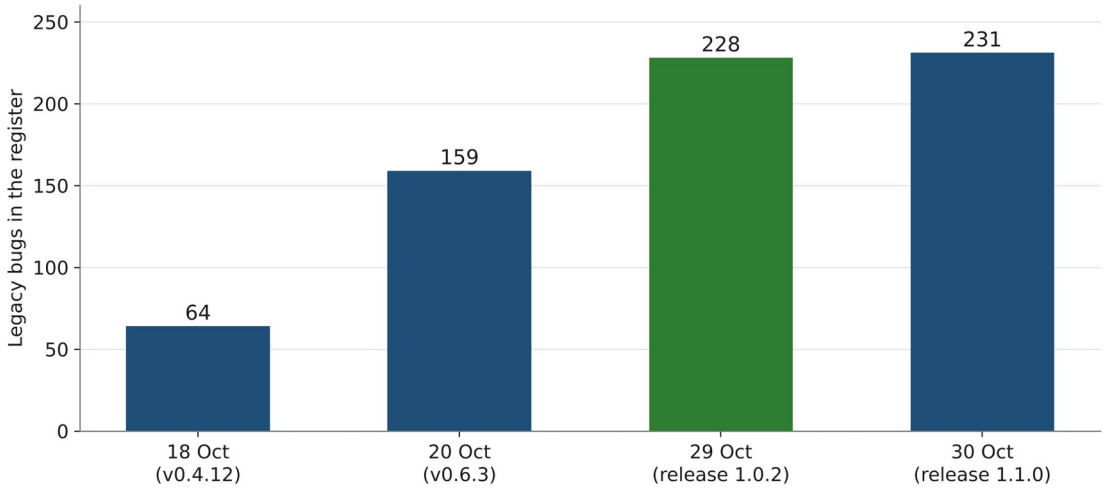
The serious bugs sit where formulas were copied across sectors or years

Code	Bug in legacy CPAT	Effect
L1	Services electricity follows industry's multiple after 2023	Largest single error in power demand
L3	Income factor on the GDP elasticity applied twice	Elasticity drifts; error grows over time
L13	Other oil products: pass-through 1, margin 0	Priced at the floor; use up about tenfold by 2030
L18	Non-energy methane never changes	A methane fee on farming and waste does nothing
L24	GDP effect added to growth every year	A one-off gain compounds
L87	Regulation lookup never matches	Regulation scenarios have no effect

Most of the 228 bugs are latent: they change nothing until an option is switched on.

Source: CPAT-AI register, release 1.0.2 (Egypt runs); working paper 5.2, What the rebuild found.

The count rose as each block was rebuilt, so 228 is a lower bound



Owners are fixing it: the CPAT team marked 63 fixed in its next file.

Checked cell by cell: 52 confirmed, 9 with caveats, 2 marked fixed that were not.

Still to come: air pollution and transport co-benefits are not yet rebuilt.

Source: CPAT-AI task log and register (October 2026).

Every number is checked by something that did not produce it

Test	Compared with	Scale at release 1.0.2
1. Method equals legacy	Legacy's cached results, bugs switched back on	Prices: 80 rows to 5.6×10^{-17}
2. Workbook equals Python	Independent recomputation; paste test	Every block, every test scenario
3. Coded CPAT equals Excel-AI	Excel-AI's stored results	7,458 codes per scenario, within 10^{-9}
4. Release equals last release	Previous release, code by code	Release 1.0.0: 0 of 15,086 changed

The tests verify; they do not validate. Validation against data is in papers 5.3 and 5.4.

Source: CPAT-AI task log, documentation and changelog; working paper 5.2, Table 2.

Building twice found seven bugs, but versions built from one reading can share an error

Python found seven bugs in Excel-AI, e.g. nominal GDP divided by a billion when already in billions; two gas price series swapped. All fixed in both.

Common-mode error is real: 27 independently written programs failed together far more than chance predicts (Knight and Leveson 1986).

AI agents too: 48 agent-written versions showed the same, though voting across three cut failures by about two thirds (Ron et al. 2026).

The guard: test 1 ties every block to legacy's own results, which no agent wrote; fixes go to the CPAT team.

Source: CPAT-AI register (M15-M22); Knight and Leveson (1986); Ron, Baudry and Monperrus (2026).

AI agents built the models; a named owner made every judgment

241 finished tasks in four weeks, through Claude Code and GitHub Copilot: builders, checks, the Python model, the register's documents.

Every departure carries a dated owner decision; the owner accepted each release.

The evidence on AI productivity is mixed: 56% faster on a set task (Peng et al. 2023); 19% slower for experienced developers on mature projects (Becker et al. 2025).

So the safeguard is the test, not the tool. Every output was checked by something its author did not write.

Source: CPAT-AI task log, 2-29 October 2026; Peng et al. (2023); Becker et al. (2025).

What to do

What the rebuild shows

228 legacy bugs, 24 serious; most latent until an option is used.

One rule makes every difference traceable.

Building twice found 7 more; it does not remove shared errors.

Agreement is not truth: validation comes next (5.3, 5.4).

Actions

Model commissioners: require a codes map, one register, a second build with regression tests, and a named owner.

CPAT team: apply the fix plans; rerun validation on the corrected model.

Bank teams running CPAT: check each run against the register before it goes into advice.

Builders using AI: agents build, people decide, tests check.