WORKING PAPER | VERSION 0.2 | OCTOBER 2026

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

What two AI-assisted rebuilds of the Climate Policy Assessment Tool found, and how policy models should be maintained

Stephen J. Stretton*

ABSTRACT

The Climate Policy Assessment Tool is a spreadsheet model that finance ministries and international institutions use to estimate what carbon pricing and related reforms do to energy use, emissions, revenue and households in more than 180 countries. Its legacy workbook is too large for anyone to audit cell by cell. This paper describes how it was rebuilt twice in October 2026, with AI coding agents doing most of the construction: once as a readable Excel workbook written by code, and once as a Python model that reproduces the workbook number for number. One rule governed both rebuilds: errors in the legacy model are fixed and never copied, while questionable but intended choices are copied and logged. The rebuild found 228 bugs in the legacy model, 24 of them serious enough to change standard results, as well as 135 issues (intended choices open to question) and 20 errors in the published documentation. Writing the model a second time found seven further bugs in the first rebuild. The lesson for ministries is that a policy model should be specified by shared variable codes, built at least twice, tested at every change, and owned by a named person who decides every departure from the original.

Keywords: climate policy modeling, carbon pricing, model verification, spreadsheet risk, N-version programming, artificial intelligence
JEL codes: C63, C88, H23, Q48, Q54

Working paper, version 0.2, October 2026. Stephen J. Stretton, writing in a personal capacity. The views expressed are the author's own and do not represent those of any institution with which he is or has been affiliated. Companion paper to the public CPAT-AI repository, https://github.com/cpmodelai/cpat_ai. A short version for policy makers is Climate Fiscal Policy Brief No. 5.2.[1]

*Writing in a personal capacity.

[1]The legacy Climate Policy Assessment Tool is the joint work of the IMF and World Bank CPAT teams; its mitigation documentation was prepared by Alexandra Campmas, Daniel Bastidas, Olivier Lelouch, Faustyna Gawryluk, Paolo Agnolucci and the author, building on earlier work by Ian Parry, Simon Black, Karlygash Zhunussova, Nate Vernon-Lin and Victor Mylonas (Campmas et al. 2023; Black et al. 2023). CPAT's air-pollution module is the work of Paulina Schulz-Antipa, who leads it in CPAT-AI; the distribution and transport co-benefit modules are led by other colleagues on the CPAT team; the mitigation rebuild and this paper are the author's. Every count in this paper is from release 1.0.2 of the repository (29 October 2026) unless stated otherwise (CPAT-AI project 2026d, 2026e). Errors are the author's own.

1 Executive summary

A climate-fiscal model that nobody can audit is a risk for every ministry that relies on it. The Climate Policy Assessment Tool (CPAT), built by the IMF and the World Bank, estimates what carbon taxes, fuel subsidy reform and related policies do to prices, fuel use, emissions, revenue, GDP and households, year by year, for more than 180 countries. It is a spreadsheet: the legacy workbook is over 21 megabytes, its main calculation sheet runs past row 18,000, and its methodology fills about 400 pages. Nobody had checked all of it against what it was meant to compute.

In October 2026 the model was rebuilt twice, with AI coding agents doing most of the construction. The first rebuild, Excel-AI, is a readable workbook written and checked by code, in which one short formula serves a whole block of cells. The second, Coded CPAT, is a Python model written from Excel-AI's equation list; it reproduces every one of Excel-AI's 7,458 stored outputs for each Egypt scenario tested, to a relative difference of one in a billion. Every input, variable and output carries one code, the same in all three models, so any two can be compared after every change.

One rule kept the rebuild faithful: bugs are never copied; issues are copied and tracked. A bug is a formula that does not compute what it means to, such as one pointing to the wrong row; it is fixed. An issue is a deliberate choice open to question, such as an old data vintage; it is reproduced exactly and logged. Any other departure needs a dated decision by the model owner, so every difference from the original has a code and a reason.

The rebuild found 228 bugs in the legacy model, 24 of them serious enough to change standard results. Services electricity demand grew with the industrial multiple after 2023, the largest single error in power demand. The GDP effect of recycled carbon revenue was added to the growth rate every year instead of to the level once, so it compounded. A methane fee on farming and waste did nothing, because non-energy methane never changed. Regulation scenarios had no effect at all. Most other bugs are latent: they fire only when a ministry asks a question the default run does not answer. The public register also lists 135 issues and 20 errors in the published documentation.

Writing the model a second time found seven bugs that the first rebuild's own checks had missed. They included nominal GDP divided by a billion when the source was already in billions, and two gas price series read from each other's rows. Building twice is an old idea (N-version programming; the UK Treasury's "parallel build"), and so is its limit: independently written versions share errors more often than chance predicts, and a 2026 experiment finds the same for versions written by AI agents.

The owners of the legacy model are fixing what the rebuild found, and the checks keep everyone honest. Within weeks the CPAT team marked 63 of the registered bugs fixed in its next file. A cell-by-cell check confirmed 52, found 9 fixed with caveats and found 2 marked fixed that were not.

AI agents changed what a small team could do, but did not make the judgments. The project log records 241 finished tasks in four weeks, and a register of 454 entries. Every decision that changes what the model means (bug or issue, which departures to make, which release to accept) stayed with a named owner. The evidence on AI productivity is mixed, and in this project every agent output was tested by something its author did not write.

Ministries and development banks should require four things of any policy model they rely on: a codes map, so that any audit or rebuild is a join; one register of bugs, issues and

departures, shared with the model's owners; a second implementation and a regression test at every release; and a named owner who decides, in writing, every departure from the original. None of this proves the model right. Three implementations can agree to nine digits and share a wrong elasticity; that is the job of validation, the subject of papers 5.3 and 5.4.

2 Introduction

A climate-fiscal model that no one can audit is a liability for the ministry that relies on it. The Climate Policy Assessment Tool (CPAT), built jointly by the International Monetary Fund and the World Bank, estimates what carbon taxes, emissions trading, fuel subsidy reform and related policies do to energy prices, fuel use, emissions, revenue, GDP and households, year by year, for more than 180 countries (CPAT Team 2023; Black et al. 2023). Legacy CPAT, the Excel model, is a binary workbook of more than 21 megabytes whose main calculation sheet runs past row 18,000 (CPAT-AI project 2026a), documented in about 400 pages (CPAT Team 2023). Its formulas differ from row to row, so checking one row says little about the next. No one had checked all of it against what it was meant to compute.

When its mitigation and distribution modules were rebuilt formula by formula in October 2026, and every difference from the original explained, the model turned out to be wrong in 228 places. Some of the errors were large. Egypt's population followed a misread growth path instead of the United Nations projection, which puts it at 147 million in 2041 (United Nations, Department of Economic and Social Affairs, Population Division 2024). The GDP effect of recycled carbon revenue was added to the growth rate every year instead of to the level once, so a one-off effect compounded. A methane fee on farming and waste had no effect at all, because non-energy methane never changed (CPAT-AI project 2026e). None of these was visible in the workbook's headline results; each needed someone to trace a formula to its source.

This paper asks whether a large spreadsheet policy model can be rebuilt so that every step can be read and tested, and what such a rebuild reveals. The answer is yes, at a cost that AI coding agents have brought within reach of a small team. CPAT-AI, the project and repository, rebuilt CPAT twice: Excel-AI, a readable workbook written and checked by code, and Coded CPAT, a Python model that reproduces Excel-AI to a relative difference of 10^{-9}. One rule governed both: **bugs are never replicated; issues are replicated and tracked.** Shared variable codes made the two rebuilds and the original comparable cell by cell. The rebuild registered 228 legacy bugs (24 of high priority), 135 issues, 35 deliberate departures and 20 errors in the published documentation; writing the model a second time in Python found seven bugs in the first rebuild.

The ideas are not new; what is new is their combination, their application to a model in active policy use, and a count of what they find. Spreadsheet audits find errors in almost every large spreadsheet (Panko 1998; Powell et al. 2008). Writing a program twice and comparing the versions is N-version programming, whose classic experiment also showed its limit: independently written versions fail together more often than independence predicts (Knight and Leveson 1986). The UK Treasury's review of government models lists "parallel build" among developer tests and asks for a named owner of each business-critical model (HM Treasury 2013). This paper adds four things. First, a replication rule that keeps a rebuild faithful where the original is right and correct where it is not, and makes every difference traceable (Section 4.2). Second, shared codes that make a second implementation cheap to compare (Section 4.3). Third, counts of what the method found in CPAT (Section 5, Section 6). Fourth, evidence on what AI coding agents did and did not change (Section 7), set against experimental evidence that points both ways (Peng et al. 2023; Becker et al. 2025). Validation against data and other models is the subject of the companion papers 5.3 and 5.4 (Stretton 2026e, 2026b); the method itself is summarized in paper 5.1 (Stretton 2026d).

3 Why spreadsheet policy models escape audit

Spreadsheets are the default platform for policy models because ministries can open them. A finance ministry economist can open CPAT, change a carbon price and see the result without installing anything. Every number is visible. That openness is real, and it is why CPAT was built in Excel: its documentation names being "spreadsheet-based with data and formulas readily accessible within the tool" as the first step taken to make CPAT transparent (CPAT Team 2023).

But visible is not the same as checkable. A spreadsheet shows each formula in isolation. The structure of the calculation (which row feeds which, which constant is used where, which sector a copied formula points to) is implicit. Panko's review concludes that errors occur in a few percent of all cells, so that for a large spreadsheet the question is how many errors it has, not whether it has any, and that testing after development matters more than development discipline (Panko 1998). In Panko's later summary of seven field audits, 94 percent of 88 operational spreadsheets had errors (Powell et al. 2008). Powell, Baker and Lawson also show how weak those estimates are: three of the seven sources are unpublished, most do not define an error, and the largest counts "issues" raised on early drafts. The firmest conclusion is theirs: no study has found operational spreadsheets free of errors.

Errors in analytical spreadsheets can change policy debates. Reinhart and Rogoff's spreadsheet left out five countries through a coding error; together with selective exclusion of data and the weighting they chose, it turned average growth of 2.2 percent in countries with public debt above 90 percent of GDP into a published −0.1 percent (Herndon et al. 2014). Economics research more broadly is hard to reproduce: Chang and Li replicated the key result of 22 of 67 papers without the authors' help and 29 of 59 with it (Chang and Li 2015).

Governments have learned this the hard way, and the remedy they adopted is organizational. Citing the failed InterCity West Coast rail franchise competition, the UK Treasury reviewed the quality assurance of government analytical models. It found just under 500 business-critical models and recommended quality assurance of each model's inputs, method and outputs, specialist staff, and a single senior owner responsible for the model through its life; among developer tests it lists "parallel build" (HM Treasury 2013). Energy-policy modelers have argued for openness on the same grounds: open models and data raise quality and reduce duplicated effort, although licensing and workload are real reasons that data stay closed (Pfenninger et al. 2017). CPAT shows both sides: its formulas are open to its users, but several of its key data tables, such as energy balances and household surveys, are licensed and cannot be published.

Legacy CPAT had the features that make errors likely and hard to find. Its formulas varied from row to row, often with long chains of lookups; constants were typed into formulas; the same quantity could appear under different names in different sheets; and the documentation and the workbook had drifted apart. Twenty errors in the published documentation were later registered, three of them serious: no equation for the GDP feedback that the workbook computes, a sign error in the electricity price markup, and a discount factor written wrongly (CPAT-AI project 2026e). A reviewer who checked the workbook against the documentation would have been checking against an imperfect standard.

The remedies are known; the labor to apply them was missing. Write the model as code, test each block, keep one register of known problems, build it twice: each is standard advice (Wilson et al. 2014; Sargent 2013; HM Treasury 2013). Applying them to a model of hundreds

of thousands of formulas without stopping the work it supports needs a software team that a policy model rarely has. This paper reports what happened when AI coding agents supplied most of that labor, under written rules that kept the human owner in charge of every judgment (Korinek 2023).

4 The design: two rebuilds and one rule

4.1 Legacy CPAT is the specification

Both rebuilds take legacy CPAT as ground truth, not the documentation and not anyone's view of what CPAT should be. The legacy workbook was kept read-only, in the three versions current during the rebuild (CPAT 1.0pre_456, 459 and 465, with the proprietary data stripped), and work was done on copies (CPAT-AI project 2026c). The project proceeds block by block in five steps: replicate a block of legacy formula by formula, with the same section numbers; check it against an independent Python recomputation and against legacy's stored results; classify every difference; release a new version only after the checks pass; and port validated blocks to Python.

The chain being replicated is the CPAT mitigation chain set out in paper 5.1 (Stretton 2026d). Policies set price signals; retail prices build up from international prices, taxes and subsidies; fuel use in 16 end-use subsectors and power responds to income, prices and efficiency; a power model dispatches plants and invests; and the results feed revenue, emissions, GDP effects and household incidence. The core is a constant-elasticity demand equation,

$$Q_{ijt} = Q_{ij,t-1}\,(1 - a_{ij})\,(1 + g_t)^{\varepsilon^Y_{ij}} \left(\frac{P_{ijt}}{P_{ij,t-1}} \right)^{\varepsilon^P_{ij}}, \tag{1}$$

for fuel i in sector j, where a_{ij} is autonomous efficiency improvement, g_t GDP growth and ε^Y, ε^P the income and price elasticities. The full form, with a separate efficiency response to prices, is in paper 5.1 and in the Total Effective Carbon Price paper (Stretton 2026c). Every block of the rebuild has to reproduce equations like equation 1 exactly as legacy computes them, except where legacy does not compute what it means to. Bug L3 is an example: the income factor on the elasticity ε^Y was applied twice and anchored to 2023, so the effective elasticity drifted as incomes rose (CPAT-AI project 2026e).

4.2 The replication rule

The rule separates two kinds of difference between what legacy does and what it should do. A *bug* is a place where legacy does not compute what it means to: a reference to the wrong row, a term applied twice, a dead or blank reference, a sign error, a formula that contradicts its own label. An *issue* is a place where legacy does what it means to, but the choice is open to question: an old data vintage, a simplification, a missing feature, a result nobody can yet explain. Bugs are fixed and never copied. Issues are copied exactly and tracked (CPAT-AI project 2026b).

Stated formally, the rule gives every difference from legacy a code. Let $L(x)$ be what legacy computes for input x, $I(x)$ what its authors intended (read from labels, documentation and the CPAT team) and $P(x)$ what the prototype computes. The rule sets $P(x) = I(x)$, except where the owner has decided a departure D. Since $I(x) = L(x)$ wherever legacy has no bug,

$$P(x) \neq L(x) \Rightarrow \text{a registered bug fix, or a registered departure decided by the model owner .} \tag{2}$$

Claim (closure). *Every difference between the prototype and legacy can be traced to a register entry, provided that three conditions hold: every difference found is registered the same day, codes are never reused, and every release is regression-tested against the previous one, so that no unregistered difference enters between releases.* The first two are rules of the project (CPAT-AI project 2026b); the third is the fourth test level in Section 6. A reviewer who finds that Excel-AI and legacy disagree on a number can look up why.

Three further rules make the closure hold in practice. When in doubt whether something is a bug or an issue, treat it as an issue, replicate it and ask. Never improve legacy silently: a better method or newer data is an issue, proposed as an upgrade for the owner to decide. And any departure that is not a bug fix needs an explicit decision by the model owner, recorded with the issue (CPAT-AI project 2026b). One example: legacy computes energy subsidies for the baseline scenario only; the prototype, by the owner's decision, computes them in every scenario, with the baseline equal to legacy (issue I110) (CPAT-AI project 2026e).

Data problems are issues, ranked by whether legacy can fix them. Where legacy reads data wrongly (a blank VAT cell read as a rate of zero), the formula fix is a legacy bug with a cell-level plan, and the data problem is an issue of priority *Very high*. Where no formula fix is possible because the data themselves are missing or poor, the prototype does exactly what legacy does and the issue is *Extremely high*. Two issues held that rank in release 1.0.2: combustion emission factors available only as base and final values, and current account data typed in for Egypt only (CPAT-AI project 2026e).

The rule also runs between the two rebuilds. Coded CPAT treats Excel-AI as its specification exactly as Excel-AI treats legacy. A bug that Python finds in the Excel prototype is never copied into Python. It is registered as a prototype bug, fixed in Python where the fix is obvious, and listed as a "known difference" that the tests check really does differ. It is then fixed in the next Excel release, after which the known difference is deleted and the two must agree again (CPAT-AI project 2026b, 2026c).

4.3 Shared codes make three models comparable

Every input, variable and output has one code, the same in legacy, Excel and Python. Inputs keep the names and rows of legacy's input sheet (`NameOfParameter`), in legacy units, so a scenario defined for one model means the same thing in the others. Outputs carry codes that state country, module, variable, sector, fuel and scenario: `egy.mit.co2.tot.tot.2` is Egypt's total CO_2 in scenario 2. A codes map lists every input with its legacy row, its name and its Python key, and Python may not invent a second name for anything that has a code (CPAT-AI project 2026b).

Shared codes turn comparison into a join. Without them, comparing two implementations of a model with 15,086 output codes means matching rows by hand. With them, a comparison is a table merge on the code and the year, followed by a tolerance check, and it can be run after every change. In the author's judgment this is the design decision that made the rest affordable: it is what let one person check three implementations against each other every day.

4.4 Excel written by code, Python written to be read

Excel-AI is a workbook that no person typed. Python builders write it with openpyxl and recalculate it in LibreOffice, so it can be built and checked on any machine (CPAT-AI project 2026c). The norms that govern it are mechanical and checkable: every formula was at most 100 characters as Excel shows it; one formula serves a whole block of cells, so that copying, dragging or pasting it anywhere in the block gives the same result; no constant is typed into a formula (only 0 and 1 are allowed); every money unit comes from one vocabulary; and every file passes a lint check before it ships, so Excel never shows a repair prompt (CPAT-AI project 2026b). The workbook of release 1.1.0 holds about half a million formulas (497,541) (CPAT-AI project 2026a).

Coded CPAT is written so that a person with the equation list open can check every equation. Each equation of the list is one small function, named after the code of the quantity it computes, whose docstring states the equation. Names are CPAT codes (`ener` for fuel use, `atp` for after-tax prices); no number other than 0 and 1 is typed into the code; unit conversions live in one place; and a test fails if any equation in the list has no function or any function cites an equation that is not in the list (CPAT-AI project 2026b). Release 1.0.2p1.1 runs one country and two scenarios in 0.8 seconds once the baseline is cached (CPAT-AI project 2026d).

5 What the rebuild found

The rebuild registered 228 legacy bugs, 24 of them of high priority, and 135 issues. Table 1 gives the classes and counts in release 1.0.2. All entries live in one register, from which every list, tracker and document is regenerated at each release; the tracker is shared with the CPAT team that maintains legacy.

Table 1. The rebuild of CPAT registered 454 entries; half are bugs in the legacy model (register of release 1.0.2, 29 October 2026)

Class	Code	Count	Breakdown	What the rebuild does
Legacy bugs	L	228	Core mitigation 149 (high 21, medium 42, low 86); power engineer model 29 (3, 10, 16); road engineer model 17 (0, 9, 8); distribution 31 (all low); inputs 1; data 1	Fixes each, or leaves the feature out; never replicates; writes a cell-level fix plan for legacy
...status in legacy 465			Fixed by the CPAT team and confirmed 52; fixed with caveats 9; marked fixed but not fixed 2; still open 165	Each claimed fix checked cell by cell
Prototype bugs	M	22	Fixed 21, withdrawn 1; 8 suspected (7 confirmed) by Coded CPAT	Fixed in Excel; never copied into Python
Documentation bugs	DOC	20	High 3, medium 7, low 10	Models follow the workbook; documentation to be corrected
Issues	I	135	Mitigation 109, data 13, distribution 9, inputs 4; major 35, minor 97, very or extremely high 3	Replicated 93; prototype limitation 16; departs by owner decision 14; out of scope 10; other 2
Departures	LD	35	Rows of the workbook's LegacyDiff sheet that are not "Same"	Kept by owner decision, each with a reason
Upgrades	U	14	High 3, medium 4, low 7	Proposed, not built

Source: CPAT-AI project (2026e) (tracker `Tracking_Legacy_Bugs_v1.0.2.xlsx`, tabs Bugs and Issues; counts recomputed by the author). *Notes:* Priority of bugs by effect on results: high, wrong results in standard runs; medium, wrong results in some runs or countries, or a wrong reported number; low, latent or reporting only. Codes are never reused, so the L series runs to L230 with 228 live entries. By release 1.1.0 (30 October 2026) the register held 231 legacy bugs, still 24 of high priority, and still 135 issues.

The high-priority bugs cluster where formulas were copied across sectors or years. The biggest affect the drivers of demand. Services electricity demand grew with the industrial multiple after 2023, because a copied formula pointed six rows up; services are about a quarter of Egypt's electricity demand, so this is the largest single error in power demand (L1). The income factor on the GDP elasticity was applied twice and anchored to 2023, so the elasticity drifted (L3). "Other oil products" carried a hardcoded pass-through and a zero margin, which priced them at the floor of 0.01 and made their use rise roughly tenfold by 2030 in the Egypt run (L13) (CPAT-AI project 2026e). A second group switches policies off without warning: regulation scenarios had no effect, because a lookup on scenario names never matched and the share it returned was zero (L87); and actual power capacity factors were never used, because the lookup searched the wrong years (L53). A third group sits in the macro-fiscal outputs: the compounding GDP effect (L24), subsidies as a share of GDP divided by the previous year's GDP (L188), and a green-investment block in which eight separate formulas read blank rows, header text or the wrong constants (L205–L213) (CPAT-AI project 2026e).

Most bugs are latent rather than visible. Over 60 percent (142 of 228) are of low priority: they affect a path that the standard Egypt run does not take, or only a reported number. The standard Egypt run, for instance, uses the simpler elasticity road model, so the 17 bugs of the road engineer model were latent in it. A latent bug is still a bug. It fires when a ministry switches on the feature it lives in, which is exactly when the ministry is asking a question the default run does not answer.

The legacy owners have begun fixing what the rebuild found. Of the bugs registered against the earlier legacy file, the CPAT team marked 63 fixed in its next file, 1.0pre_465. A cell-by-cell check confirmed 52, found 9 fixed with caveats, and found 2 marked fixed that were not (CPAT-AI project 2026e). That last number is a small argument for the whole approach: even careful owners fixing a known list need an independent check.

The 135 issues are mostly questions for the modelers, not errors. They range from data limitations to design choices. Egypt's greenhouse gas inventory breaks after 2021, so calibrating to 2024, as legacy does, may understate energy methane by about 19 Mt CO_2e (I91); the distribution module always uses the mean elasticity, because the data hold no other statistic (I96) (CPAT-AI project 2026e). Ninety-three issues are replicated exactly as legacy has them. Fourteen depart from legacy by explicit owner decision, such as computing subsidies in every scenario (I110). The register keeps the questions visible without letting the rebuild answer them unilaterally.

The documentation needed correcting too. The 20 documentation bugs include equations written with the wrong sign, an undefined symbol, a derivative computed wrongly, and units that differ between notation tables. None affects the models, because the models follow the workbook, but each would mislead a reader who used the documentation to check the workbook (CPAT-AI project 2026e).

The count rose as the rebuild went deeper, so 228 is a lower bound. After version 0.4.12 of the prototype, on 18 October 2026, the register held 64 legacy bugs; at version 0.6.3, two days later, 159 (128 in mitigation and 31 in distribution); at release 1.0.2, on 29 October, 228; and at release 1.1.0, a day later, 231 (CPAT-AI project 2026a, 2026e). The jumps came as new blocks were rebuilt (distribution, revenue, welfare, green investment) and as the CPAT team's own issue lists were mapped onto the register. The modules not yet rebuilt (air pollution, transport co-benefits) will add more.

6 The testing regime

Every number is checked by something that did not produce it. The regime has four levels (Table 2). Each answers a different question, and none alone would be enough.

Table 2. Four levels of testing: each checks a number against something that did not produce it

Level	Question	Compared with	Tolerance	Scale at release 1.0.2
1. Method equals legacy	Did the rebuild understand legacy before fixing it?	Legacy's own cached results, with every bug switched back on	Precision of the cache	Prices 80 rows (largest gap 5.6×10^{-17}); power 208 rows; road engineer model about 1,091 series (4×10^{-14}); distribution (4×10^{-4})
2. Workbook equals Python reference	Does the workbook compute what the corrected method says?	Independent Python recomputation of each block; property tests; paste test	Exact or near-exact	Every block, every test scenario
3. Coded CPAT equals Excel-AI	Do two implementations agree?	Excel-AI's stored results	Relative 10^{-9}	Egypt, scenarios 1 and 2, 2022–2041: 7,458 output codes; distribution 1,684 codes; stored scenarios 3–6: 123 key codes each
4. Release equals last release	Did anything change that should not have?	The previous release, code by code	Zero, or explained in the version table	Release 1.0.0: 0 of 15,086 codes changed

Sources: CPAT-AI project (2026a); CPAT-AI project (2026c); CPAT-AI project (2026d).

First, method equals legacy. For each block, a Python reference implementation is fed legacy's own cached inputs with every bug switched back on, and must reproduce legacy's stored results. The price block reproduced 80 legacy rows with a largest relative difference of 5.6×10^{-17}; the power block's reference reproduces 208 legacy rows of the baseline and a carbon-tax run; the road engineer model reproduces legacy's cached run (both scenarios, about 1,091 series, 2022–2041) to about 4×10^{-14}; the distribution reference reproduces legacy's effects and recycling to 4×10^{-4}, the rounding of the cache (CPAT-AI project 2026a). This step proves that the rebuild understood the method before fixing anything in it, and it is the one level that ties the work to numbers no agent wrote.

Second, workbook equals Python. With the bugs fixed and the prototype's own inputs, every row of the workbook must equal the Python reference in every test scenario. Further tests check properties that must hold whatever the numbers: generation equals required generation, the capacity balance holds, a carbon price raises the share of variable renewables and lowers power CO_2, and totals equal the sum of their parts. A paste test copies a

scenario's block of columns, as a user would, and checks that the copies number themselves and reproduce the stored results (CPAT-AI project 2026c). Some power rows added in release 0.9.0 still have no independent recomputation; the full check lists them, and they are covered by level 3 (CPAT-AI project 2026a).

Third, Coded CPAT equals Excel-AI. For Egypt, scenarios 1 and 2, every year 2022–2041, each of the 7,472 reference output codes of Excel-AI's stored results (7,458 with values) must be matched by Coded CPAT, for every code k and year t,

$$\frac{\left|y_{kt}^{\text{Python}} - y_{kt}^{\text{Excel}}\right|}{\max\left(1, \left|y_{kt}^{\text{Excel}}\right|\right)} \leq 10^{-9}, \tag{3}$$

except for codes listed as known differences, which must differ. The distribution module is compared the same way on 1,684 codes, and stored scenarios 3–6 on their 123 key codes each. The Python suite of release 1.0.2p1.1 has 65 tests and runs in about a minute; it must pass before every commit, and a failing comparison may not be loosened to pass (CPAT-AI project 2026d, 2026b).

Fourth, every release is a regression test. Each new version's outputs are compared, code by code, with the previous release, and every difference must be zero or explained in the version table. Release 1.0.0 changed 0 of 15,086 output codes against its predecessor (CPAT-AI project 2026a). When a release does change results, the version table names the register codes responsible: release 0.9.2, for instance, changed only nominal GDP in the mitigation results, because it fixed a unit error (M19) that Python had found (CPAT-AI project 2026d).

Writing the model twice is itself a test, and it found bugs that the other checks missed. When Coded CPAT was written from the equation list and Excel-AI, its comparisons flagged eight places where the prototype did not compute what it meant to: VAT on efficient prices at the 2022 rate in every year (M15); a freight electric-vehicle feebate valued at the combustion-engine truck's distance (M16); a plug-in hybrid cost that mixed electric capital cost with combustion insurance and maintenance (M17); nominal GDP divided by a billion although the source is already in billions (M19); two international gas price series read from each other's rows (M20); a division by zero in the methane fee where a curve exponent is zero (M21); and a distribution row that added a percentage to amounts in local currency (M22) (CPAT-AI project 2026e). The eighth (M18) was withdrawn on review. The other seven were fixed in Excel-AI 0.9.2 and in Python, with no known differences left. The checks inside the Excel build had not caught them.

What the tests prove, and what they do not. The regime verifies; it does not validate (Sargent 2013). It shows that three implementations compute the same thing and that every difference from legacy is a registered decision. It does not show that the elasticities are right, that the baseline fits history, or that the power model agrees with a full capacity-expansion model. Those are questions about the model, not the code, and they are taken up in papers 5.3 and 5.4 (Stretton 2026e, 2026b). The first systematic comparison of Coded CPAT with legacy's cached Egypt baseline shows the split. In the base year 2022, all 15 CO_2 rows and 91 percent of 129 energy-use rows are within 5 percent of legacy, because both start from the same energy balance; only 48 percent of 128 price rows are, because the rebuild uses price data two years newer and fixes the legacy price bugs (CPAT-AI project 2026c).

7 What AI assistance did and did not do

AI coding agents did most of the construction. The repository is organized so that any AI tool starts from one routing file (`AGENTS.md`), which sends it to the same norms, bug lists, to-do lists and task log that people use; to change how the AI behaves, the team changes the norms, not the AI's own instructions (CPAT-AI project 2026d, 2026b). Working under those norms, through Claude Code and GitHub Copilot, agents wrote the Python builders that generate the Excel workbook, the independent Python recomputations that check it, the whole of Coded CPAT and its tests, and the generators that turn the one register into the bug document, the tracker, the short lists and the documentation website. They surveyed legacy formula dumps block by block and drafted the cell-level fix plans that the CPAT team can apply to legacy (CPAT-AI project 2026a).

The pace changed what was feasible for one owner. The task log records 241 finished tasks between 2 and 29 October 2026, each with its inputs, outputs and caveats (CPAT-AI project 2026a). In four weeks the project went from a kernel workbook to a released, replicated model with a register of 454 entries. The author's judgment is that a rebuild by hand on this scale needs a software team for many months, which a policy model maintained by economists does not have; that is one reason legacy CPAT had never been fully audited. The experimental evidence on AI productivity does not settle the counterfactual. In a controlled task, developers with an AI pair programmer finished 55.8 percent faster (Peng et al. 2023). In a randomized trial with experienced developers working on their own mature projects, early-2025 AI tools made them 19 percent slower, although they expected to be 24 percent faster and afterward believed they had been (Becker et al. 2025). CPAT-AI is closer to the first setting at the start (new code written to a clear specification) and closer to the second later (a large codebase with many rules). The log shows output, not a controlled comparison.

The agents did not make the judgments. Whether a difference is a bug or an issue is decided by rule, and the rule's default is "issue: replicate it and ask". Every departure from legacy that is not a bug fix carries an explicit decision by the model owner, dated in the register; a typical entry reads "user decision 2026-10-29: legacy prices". The choice of scope (which modules to build, which upgrades to defer), the treatment of data, and acceptance of each release were the owner's. Review in Excel 365, the environment in which ministries use the workbook, was still the owner's open task at release 1.0.2 (issue I1) (CPAT-AI project 2026e).

The agents' work needed the same checks as anyone's, and so did the people's. The task log records where AI output was taken partly on trust and says so: an equation list drafted by a subagent was spot-checked rather than checked line by line (CPAT-AI project 2026a). The checks caught agent errors as they were meant to; a full check, for instance, stopped a release over one formula of 114 characters. And the same checks found two legacy bugs that the human owners had marked fixed but had not fixed (CPAT-AI project 2026e). The testing regime in Section 6 exists because neither the agents nor the people can be trusted to be right without an independent test. A further norm makes the regime learn: a mistake whose cause could recur ends with a new norm and a check that would have caught it, added in the same task (CPAT-AI project 2026b).

The strongest objection is common-mode error, and the evidence says it is real. If the same kind of agent writes the Excel builder, the Python recomputation and Coded CPAT from the same reading of legacy, a misreading can be copied into all three, and the tolerance test in equation 3 would pass. Software engineering has known this for forty years. In Knight and Leveson's experiment, 27 versions of one program written independently

from the same specification were each highly reliable, but failed together far more often than independence predicts (Knight and Leveson 1986). A 2026 replication with 48 versions written by AI coding agents again finds substantial common-mode failure, much of it where the specification is ambiguous; voting across three versions still cut the mean number of failures from 387 to 131 (Ron et al. 2026). Building twice therefore helps, and the seven prototype bugs show it, but the second build is not independent of the first: Coded CPAT was written from Excel-AI's equation list and checked against its stored values, so it shares Excel-AI's reading of legacy by construction.

Three features limit the common-mode risk without removing it. The first test level ties each block back to legacy's own stored results, which no agent wrote, with legacy's bugs switched back on; a shared misreading of a formula would fail there. External validation against data and other models, in papers 5.3 and 5.4, tests the whole chain against things outside it (Stretton 2026e, 2026b). And the fix plans and the tracker go to the CPAT team, whose members wrote legacy. The residual risk is a shared misunderstanding of what legacy *intends*, where the specification itself is ambiguous; only review by the model's owners and users can catch that.

8 Lessons for building and maintaining policy models

Specify the model by codes, not by cell addresses. A shared code for every input, variable and output is what made three implementations comparable after every change. Ministries commissioning a model should require a codes map as a deliverable, so that any later reimplementation, audit or comparison is a table join rather than an archaeology project. The cost is a naming protocol agreed at the start; the failure condition is a model whose owners let names drift between versions.

Keep one register of everything known to be wrong or open. Bugs, issues, departures and upgrades belong in one list with one set of fields, from which every document is generated. A register shared with the model's owners turns an audit into a work plan: the CPAT team marked 63 legacy bugs fixed in its next file within weeks of receiving the register, and each legacy bug comes with a cell-level fix plan where one is possible. The cost is discipline at every task; the failure condition is a register that is not regenerated, so that documents and code disagree.

Copy issues, fix bugs, and decide departures in writing. The replication rule keeps a rebuild faithful where the original is right and correct where it is not, and makes every difference traceable (equation 2). Its value is that it stops a rebuild from becoming a new model by accident. It costs owner time: someone with authority has to decide each departure, which is the single senior owner the UK review asks for (HM Treasury 2013). It fails if the boundary between bug and issue is drawn by whoever happens to be coding.

Build the model twice, and test every change against the last release. A second implementation in a different language finds errors that checks within one implementation miss. Regression tests on every output code make each release a controlled experiment. The cost, once AI agents do most of the writing, is modest: Coded CPAT replicated Excel-AI's mitigation and distribution modules within the same month. The failure condition is common-mode error (Knight and Leveson 1986; Ron et al. 2026), which only a test against the original's own results, external validation and human review address.

Write the methodology for a reader who never saw an earlier version, and keep change history elsewhere. CPAT-AI separates method notes, which describe the current method

only, from version notes and the task log (CPAT-AI project 2026b). Legacy CPAT's documentation had drifted from the workbook partly because the two jobs were mixed.

Use AI agents as builders under written norms, not as authorities. The agents in this project wrote code, documents and plans at a pace no single owner could match, but every judgment that changes what the model means was kept with a named person, and every output was tested by something independent of its author. That division of labor is the transferable lesson. It applies to any ministry or institution that maintains a model it did not build and cannot fully read.

The approach has limits a ministry should know. Excel-AI and Coded CPAT shipped at release 1.0.2 with complete data for Egypt only, because CPAT's licensed data tables (energy balances, household surveys and others) were held for Egypt only in the legacy file the project used; other countries need the full database (issue I90) (CPAT-AI project 2026e). Since release 1.1.0 the public repository carries those tables only in perturbed form (CPAT-AI project 2026d). The air-pollution and transport co-benefit modules are not yet rebuilt. The rebuild also inherits a structural problem it does not fix: CPAT describes road transport three times, in a simple fuel equation, an engineering fleet model and a co-benefits module that values congestion, accidents and road damage, and the three are not consistent. The co-benefits module always takes vehicle-km from the fleet model, even when fuel use and emissions follow the fuel equation, so the change in driving it values need not match the modeled change in fuel use (issue I100) (CPAT-AI project 2026e; Stretton 2026a). And the whole regime proves consistency, not truth: a faithfully replicated wrong elasticity is still wrong.

9 Conclusion

Legacy CPAT was a working model that nobody could fully audit. Rebuilt twice under a rule that fixes bugs and copies issues, with shared codes and tests on every output, it turned out to contain 228 bugs, 24 of which change standard results, alongside 135 issues and 20 errors in its documentation; the second rebuild found seven bugs in the first. What is now known is not only that the model had errors, which was likely, but where each one is, how large it is and how to fix it in the original, in a register the owners are already working through. The main limitation is that verification is not validation: three implementations that agree to nine digits can share an assumption that the world does not. The next step is to rerun CPAT's published validation (Stretton 2026e) and its elasticity checks (Stretton 2026b) on the corrected model for every country, once the full data are in, so that the model a ministry uses is both the model its authors intended and one that fits the evidence.

References

Becker, Joel, Nate Rush, Elizabeth Barnes, and David Rein. 2025. *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity*. arXiv:2507.09089 [cs.AI]. https://doi.org/10.48550/arXiv.2507.09089.

Black, Simon, Ian Parry, Victor Mylonas, Nate Vernon-Lin, and Karlygash Zhunussova. 2023. *The IMF-World Bank Climate Policy Assessment Tool (CPAT): A Model to Help Countries Mitigate Climate Change*. IMF Working Paper 2023/128. International Monetary Fund. https://doi.org/10.5089/9798400242342.001.

Campmas, Alexandra, Daniel Bastidas, Olivier Lelouch, Faustyna Gawryluk, Paolo Agnolucci, and Stephen Stretton. 2023. "The Mitigation Module." In *CPAT Documentation: WB-IMF Climate Policy Assessment Tool*. World Bank; International Monetary Fund. https://cpmodelai.github.io/cpat_ai/.

Chang, Andrew C., and Phillip Li. 2015. *Is Economics Research Replicable? Sixty Published Papers from Thirteen Journals Say "Usually Not"*. Finance and Economics Discussion Series Nos. 2015-083. Board of Governors of the Federal Reserve System. https://doi.org/10.17016/FEDS.2015.083.

CPAT Team. 2023. *CPAT Documentation: WB-IMF Climate Policy Assessment Tool*. World Bank; International Monetary Fund. https://cpmodelai.github.io/cpat_ai/.

CPAT-AI project. 2026a. *CAVEATS: Log of Completed Tasks*. CPAT-AI repository, excel_ai/core/CAVEATS.md (entries 2 to 31 October 2026). https://github.com/cpmodelai/cpat_ai/blob/main/excel_ai/core/CAVEATS.md.

CPAT-AI project. 2026b. *CPAT-AI Norms (Excel-AI) and the Python Norms of Coded CPAT*. CPAT-AI repository, excel_ai/NORMS.md (now excel_ai/core/NORMS.md) and coded/NORMS.md. https://github.com/cpmodelai/cpat_ai/blob/main/excel_ai/core/NORMS.md.

CPAT-AI project. 2026c. *CPAT-AI Project Documentation and Coded CPAT Documentation (Testing, Architecture), with the Legacy Comparison*. CPAT-AI repository, docs/CPAT_ai_documentation_qmd, docs/CPAT_coded_documentation_qmd and coded/cpat_testing/compare/legacy_comparison.md; website https://cpmodelai.github.io/cpat_ai/. https://cpmodelai.github.io/cpat_ai/.

CPAT-AI project. 2026d. *CPAT-AI: Excel-AI and Coded CPAT, Release 1.0.2 (Excel-AI) and 1.0.2p1.1 (Coded CPAT)*. GitHub repository cpmodelai/cpat_ai, releases of 29 October 2026; README.md, coded/README.md and coded/CHANGELOG.md. https://github.com/cpmodelai/cpat_ai.

CPAT-AI project. 2026e. *Tracking Legacy Bugs, Version 1.0.2, and Legacy CPAT Bugs, Version 1.0.2: The Register of Bugs, Issues, Departures and Upgrades*. CPAT-AI repository, excel_ai/bugs_and_legacy/register/releases/Tracking_Legacy_Bugs_v1.0.2.xlsx and Legacy_CPAT_Bugs_v1.0.2.docx (29 October 2026); current lists in excel_ai/core/BUGS.md and ISSUES.md. https://github.com/cpmodelai/cpat_ai/tree/main/excel_ai/bugs_and_legacy.

Herndon, Thomas, Michael Ash, and Robert Pollin. 2014. "Does High Public Debt Consistently Stifle Economic Growth? A Critique of Reinhart and Rogoff." *Cambridge Journal of Economics* 38 (2): 257–79. https://doi.org/10.1093/cje/bet075.

HM Treasury. 2013. *Review of Quality Assurance of Government Analytical Models: Final Report*. Report by Sir Nicholas Macpherson. HM Treasury. https://www.gov.uk/government/publications/review-of-quality-assurance-of-government-models.

Knight, John C., and Nancy G. Leveson. 1986. "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming." *IEEE Transactions on Software Engineering* SE-12 (1): 96–109. https://doi.org/10.1109/TSE.1986.6312924.

Korinek, Anton. 2023. "Generative AI for Economic Research: Use Cases and Implications for Economists." *Journal of Economic Literature* 61 (4): 1281–317. https://doi.org/10.1257/jel.20231736.

Panko, Raymond R. 1998. "What We Know about Spreadsheet Errors." *Journal of End User Computing* 10 (2): 15–21. https://doi.org/10.4018/joeuc.1998040102.

Peng, Sida, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot*. arXiv:2302.06590 [cs.SE]. https://doi.org/10.48550/arXiv.2302.06590.

Pfenninger, Stefan, Joseph DeCarolis, Lion Hirth, Sylvain Quoilin, and Iain Staffell. 2017. "The Importance of Open Data and Software: Is Energy Research Lagging Behind?" *Energy Policy* 101: 211–15. https://doi.org/10.1016/j.enpol.2016.11.046.

Powell, Stephen G., Kenneth R. Baker, and Barry Lawson. 2008. "A Critical Review of the Literature on Spreadsheet Errors." *Decision Support Systems* 46 (1): 128–38. https://doi.org/10.1016/j.dss.2008.06.001.

Ron, Javier, Benoit Baudry, and Martin Monperrus. 2026. *N-Version Programming with Coding Agents*. arXiv:2606.20158 [cs.SE], 18 June 2026. https://doi.org/10.48550/arXiv.2606.20158.

Sargent, Robert G. 2013. "Verification and Validation of Simulation Models." *Journal of Simulation* 7 (1): 12–24. https://doi.org/10.1057/jos.2012.20.

Stretton, Stephen J. 2026a. "Activity Levels Across CPAT, SISEPUEDE, CBAM and FTT." Unpublished manuscript.

Stretton, Stephen J. 2026b. "Are CPAT's Price Elasticities Right?" Unpublished manuscript.

Stretton, Stephen J. 2026c. *Measure Carbon Pricing by the Emissions It Cuts: The Total Effective Carbon Price*. 6.1. Climate Fiscal Policy Briefs.

Stretton, Stephen J. 2026d. "The CPAT Mitigation Module: Methods in Brief." Unpublished manuscript.

Stretton, Stephen J. 2026e. "Validating CPAT: Baselines, Back-Casts and Model Comparisons." Unpublished manuscript.

United Nations, Department of Economic and Social Affairs, Population Division. 2024. *World Population Prospects 2024*. Medium variant, total population by sex (WPP2024_TotalPopulationBySex.csv). https://population.un.org/wpp/.

Wilson, Greg, D. A. Aruliah, C. Titus Brown, et al. 2014. "Best Practices for Scientific Computing." *PLoS Biology* 12 (1): e1001745. https://doi.org/10.1371/journal.pbio.1001745.