RFI Response: AI Tools to Streamline Public-Sector Procedures

Recoding America Fund

Max Ghenis, CEO
max@policyengine.org
policyengine.org

1. About PolicyEngine

PolicyEngine is a fiscally sponsored project of the PSL Foundation, a 501(c)(3) nonprofit. We build open-source infrastructure for "computable policy"—executable representations of tax and benefit law that can be analyzed, simulated, and optimized. We translate legislation into code, enabling precise calculation of how laws affect households and government budgets across all 50 states.

We have encoded over 9,000 parameters and variables across federal and state tax-benefit systems, covering income taxes, SNAP, Medicaid, TANF, EITC, and dozens of other programs in all 50 states. Every calculation traces to one of 1,800+ statutory citations linking our code to authoritative sources in the U.S. Code, CFR, and state statutes.

Government users include the Joint Economic Committee, the UK Cabinet Office and HM Treasury, and the New York State Legislature. We are an NSF POSE Phase I awardee (#2229069) for open-source ecosystem development. Our calculations are validated against NBER's TAXSIM (with an MOU and advisory relationship with Dan Feenberg, TAXSIM's creator) and the Atlanta Fed Policy Rules Database for benefit program parameters.

Our platform already does what the RFI describes: we scan statutes and regulations, compile them into executable logic, and surface inconsistencies, gaps, and complexity. We use agentic AI workflows to translate statutory text into validated Python logic. We propose extending this capability into a comprehensive diagnostic and reform toolkit for state governments.

2. Our Perspective on RAF's Concepts

The Core Insight: Law is Code

The RFI asks for tools that "scan a state's statutory codes for sources of burden." Our experience suggests that **text scanning alone is insufficient**. Laws are inherently computational—they define inputs, conditions, and outputs. To truly identify burden, you must **compile the law into executable logic** and then analyze that logic for complexity.

This is not a metaphor. When a statute says "if income exceeds 130% of the federal poverty level, benefits shall be reduced by 30 cents for each dollar above that threshold," it is describing an algorithm. Our insight is that by making this algorithm explicit—by compiling law into executable code—we gain powerful analytical capabilities that text analysis cannot provide.

When we encode a statute as Python, we can mathematically identify circular dependencies (rules that reference each other in ways that create infinite loops), logical inconsistencies (where regulations contradict their parent statutes), and dead code (provisions that are technically on the books but mathematically impossible to trigger). We can compute complexity metrics—the number of decision branches, input requirements, and computational steps required to determine eligibility. We can detect cliff effects where small changes in circumstances cause disproportionately large changes in outcomes, and information redundancy where multiple provisions require the same underlying data.

> This approach transforms "burden identification" from a subjective reading exercise into a **quantitative diagnostic**. We can measure complexity in bits, count decision branches, and compare states objectively.

Our Proposed Tools

Tool 1: AI-Powered Statute-to-Code Compiler (RFI Concepts 1, 2) — We propose extending our existing agentic AI pipeline to ingest any state's statutory and administrative codes. The pipeline takes raw legislative text—statutes, regulations, and guidance documents—and uses large language models to generate Python code, with human experts validating edge cases. The output is executable policy logic with full citation traceability to the original statutory language. Once compiled, our engine can automatically flag outdated provisions (such as dollar thresholds not indexed for inflation since 1990), conflicting rules (two statutes defining the same term differently), and duplicative requirements (multiple forms collecting the same information). It can also identify statutory reporting and meeting requirements, wet signature mandates, and notary requirements.

Tool 2: Complexity Analyzer (RFI Concept 4) — For each policy domain, we generate a "complexity score" based on the number of inputs required from applicants, the number of decision branches in eligibility logic, the frequency of cross-references to other statutes, and the presence of cliff effects. This allows states to identify which programs impose the highest procedural burden per applicant and prioritize reform efforts accordingly.

Tool 3: Cross-State Benchmarking (RFI Concept 5a) — Because we already encode policy for all 50 states, we can immediately answer questions like: How does State X's SNAP asset test compare to other states? Which states require in-person interviews for Medicaid? What is the median complexity score for TANF eligibility? This cross-state analysis surfaces "positive deviants"—states that achieve the same policy goals with simpler procedures—and provides ready-made templates for reform.

Tool 4: Plain-Language Rewriter (RFI Concept 3) — Our platform already generates human-readable explanations of policy logic. We propose extending this capability to rewrite regulations in plain language while preserving legal requirements, generate applicant-facing guidance that accurately reflects the underlying rules, and flag passages where the original text is vague or ambiguous.

Tool 5: Model Legislation Generator (RFI Concept 5b-c) — When we identify a burdensome provision, we can draft specific statutory amendments to eliminate the burden, alternative regulatory language achieving the same policy goal with fewer steps, and comprehensive "deproceduralization packages" addressing multiple related burdens.

3. Technical Architecture

Our core stack consists of policyengine-core, a Python-based declarative policy specification engine where rules are encoded with embedded metadata, citations, and period-aware logic. Time-varying policy values (rates, thresholds, brackets) are stored in YAML files separately from formulas, with full legislative provenance. We perform compile-time analysis of rule relationships through dependency graph construction, enabling automated conflict detection. For statute parsing and code generation, we use Claude and GPT-4 in agentic workflows with human-in-the-loop validation, with continuous automated comparison against TAXSIM and Atlanta Fed benchmarks.

Our Approach to AI-Generated Rules

A key technical challenge is ensuring AI-generated code correctly implements statutory intent. Our approach relies on constrained generation: the AI generates code conforming to our domain-specific structure (variables, parameters, entities, periods), not arbitrary Python. We use citation-first design where every generated rule must link to specific statutory sections—citations are syntax, not comments. Test-driven validation means the AI generates test cases from statutory examples while human experts validate edge cases. Finally, formal dependency analysis lets us build a complete dependency graph once code is compiled, enabling mathematical verification of properties like the absence of cycles, type safety, and period compatibility.

This "AI-native" design makes it safer to use AI for rule generation—the constrained output format limits what can go wrong.

Key Technical Challenges and Solutions

We address several technical challenges in this work. For context window limits on long documents, we use chunking with citation-aware boundaries and retrieval-augmented generation. To verify correct statutory interpretation, we employ test-driven development with known outcomes, human expert review, and validation against public datasets including TAXSIM and the Atlanta Fed Policy Rules Database. To demonstrate success to reviewers, every output includes source citations with diff views showing before/after changes and quantitative complexity metrics. For handling taxpayer elections and scenario branching, we use optimized scenario evaluation for provisions requiring choice (such as itemization versus standard deduction, or filing status selection).

Data Requirements from States: We would need access to full statutory and administrative codes (typically public), agency guidance documents and policy manuals, and historical data on application denials, processing times, and appeals (for validation).

4. Partnership Model

PolicyEngine is fully aligned with RAF's co-development and open-source requirements.

All tools developed for this project will be released under AGPL or MIT licenses. Any state can fork, adapt, and deploy our code at no cost. We actively maintain public repositories and welcome contributions.

We view states as co-developers, not customers. We need state input to validate our diagnostic outputs against administrative reality, and we provide states with actionable reform recommendations, not just raw analysis.

While PolicyEngine focuses on the statutory/regulatory layer, we partner with organizations working at the applicant interface. MyFriendBen, for example, deploys our rules engine to power benefits screening tools. This creates a feedback loop: they surface "friction hotspots" from real applicant behavior, which we trace back to specific statutory provisions, enabling targeted reform.

5. Indicative Cost and Timeline

We present two scopes: a focused pilot and a more ambitious comprehensive engagement.

Option A: Focused Pilot (12 months, $350,000)

Phase	Activities	Timeline	Cost
Phase 1: Ingest	AI-assisted encoding of state statutes + regulations for 3-5 priority policy domains (e.g., Medicaid eligibility, TANF, occupational licensing)	Months 1-4	$150,000
Phase 2: Diagnose	Run complexity analysis; generate burden audit identifying top procedural friction points	Months 5-8	$100,000
Phase 3: Reform	Draft model legislation and regulatory amendments; build deproceduralization dashboard	Months 9-12	$100,000

Option B: Comprehensive Engagement (18 months, $600,000-750,000)

For states seeking deeper transformation:

Phase	Activities	Timeline	Cost
Phase 1: Full Ingest	Comprehensive encoding of all major benefit programs + licensing/permitting regimes	Months 1-6	$250,000
Phase 2: Deep Diagnosis	Complexity analysis + cross-state benchmarking + "what-if" reform simulation	Months 7-12	$200,000
Phase 3: Reform + Monitoring	Model legislation; regulatory packages; ongoing burden monitoring dashboard; staff training	Months 13-18	$150,000-300,000

The comprehensive option enables the state to maintain an ongoing "burden budget"—a real-time view of procedural complexity across agencies with alerts when new regulations exceed thresholds.

Scaling to Additional States

Because ~60-80% of policy logic is federal or follows common patterns, subsequent states cost significantly less:

Scenario	Estimated Cost	Timeline
State 2 (focused)	$150,000-200,000	6-8 months
State 3+ (focused)	$100,000-150,000	4-6 months
At scale (10+ states)	$75,000-100,000	3-4 months

The marginal cost of adding states decreases substantially because the core infrastructure—federal statute encoding, AI pipelines, diagnostic tools—is reusable.

6. Why PolicyEngine

We've already done this. We have 9,000+ parameters and variables encoded with 1,800+ statutory citations. This isn't a research project—it's an extension of production infrastructure that governments already use.

Open source is in our DNA. We don't sell software licenses. Our business model is grants and contracts for public-interest work. Everything we build is freely available. This aligns perfectly with RAF's interest in tools that can scale to multiple states at low cost.

We are cross-state by default. We already cover all 50 states, enabling immediate benchmarking without building state-specific tools from scratch. "How does your SNAP application process compare to the 10 least-burdensome states?" is a question we can answer today.

Our AI approach is bidirectional: we use AI to generate code from statutes and to generate plain-language explanations from code. This powers both diagnostic tools (finding burden) and reform tools (drafting simpler alternatives).

We have continuous validation infrastructure. Our platform includes automated test suites comparing computed outcomes to TAXSIM and Atlanta Fed benchmarks. When we encode a state's rules, we know the implementation is correct.

We have a government track record. The Joint Economic Committee uses our infrastructure for policy analysis. The UK Cabinet Office and HM Treasury have seconded staff to work with our platform. We understand how to work with government partners.

7. Additional Ideas

Beyond the RFI's listed concepts, we see opportunities for **regulatory impact simulation**: before a state adopts a procedural reform, we could simulate its effects on caseloads, processing times, and budget. Our microsimulation engine can estimate how many applicants would be affected by removing an asset test, for example.

We could also build **"complexity budget" dashboards**, giving state leadership a real-time view of procedural burden across agencies with alerts when new regulations exceed complexity thresholds.

Finally, **citizen-facing transparency**: publishing simplified, accurate explanations of every state program, generated directly from the underlying code. This reduces burden on call centers and helps applicants understand their options.

8. Conclusion

PolicyEngine offers RAF a partner that has already built the core infrastructure for computable policy. We're ready to extend our platform into a comprehensive diagnostic and reform toolkit—one that is open source, cross-state by design, and grounded in the mathematical reality of how laws actually work.

We welcome the opportunity to co-develop these tools with RAF and participating states.

Max Ghenis, CEO
PolicyEngine (a fiscally sponsored project of the PSL Foundation)
max@policyengine.org · policyengine.org

POLICY ENGINE