# RaC State of Art

Based on [2024-12-PERRON-LOGIE.pdf](https://014gc.sharepoint.com/:b:/r/sites/AICoE/Shared%20Documents/AI%20COE/EVA%20Foundational%20Project/Backlog/2024-12-PERRON-LOGIE.pdf?csf=1&web=1&e=o6TVeb)

The **current state of the art** for the "Rules as Code" (RaC) initiative in Canada—especially within ESDC and BDM Innovation—is well represented by recent experimentation with **Blawx**, a logic-based, open-source tool designed to encode legislation into machine-readable rules. The standout example, covered in the article *Rules as Code vs. ChatGPT* (Perron & Logie, Dec 2024), involves encoding the *Navigable Waters Act* and comparing the results with ChatGPT outputs. Here’s a breakdown of what’s happening and what’s next:

### **✅ State of the Art (2024–2025)**

1. **Symbolic AI Using Blawx (s(CASP) engine)**
   1. Legal rules are encoded directly by lawyers and legislative drafters using a drag-and-drop Blockly interface.
   2. Produces **explainable**, **traceable**, and **logically consistent** outputs.
   3. Allows "why" and "why not" reasoning—important for legislative clarity and transparency.
   4. Proven to outperform ChatGPT (even GPT-4o) in accuracy when applied to legislation like the *Navigable Waters Act*.
2. **Comparison with Generative AI**
   1. ChatGPT and other LLMs hallucinated legislative references and gave inconsistent or legally flawed outputs.
   2. Generative AI lacks transparency ("black box") and explainability, which poses a rule-of-law risk in legal contexts.
3. **Hybrid Neuro-Symbolic Approach**
   1. Blawx can produce structured logical outputs.
   2. ChatGPT can then be prompted with those outputs to generate **plain language summaries**.
   3. Early tests of this hybrid approach have been successful, maintaining legal accuracy while improving readability for end-users (e.g., for chatbot interfaces).

### **🔭 Where It’s Going**

1. **Increased Adoption of Blawx & Similar Tools**
   1. Canada School of Public Service (CSPS) and Justice Canada are actively piloting this approach.
   2. Legislative encoding will likely become standard in policy simulation, legal chatbot backends, and compliance auditing.
2. **Generative AI + Rules as Code Integration**
   1. Projects are underway to build conversational chatbots powered by RaC encodings.
   2. Upcoming AI models like **Logic-LM** and tools by Jason Morris (Blawx creator) aim to automate initial drafts of legal code, later reviewed by legal experts.
3. **Scalable Policy Automation**
   1. ESDC and other departments are exploring RaC to:
      1. Improve service delivery and reduce development time of operational systems.
      2. Support agile changes to benefits and regulations via logic-driven systems.
      3. Enable risk analysis and visualization (e.g., simulation of policy impacts).
4. **Standardization of Machine-Readable Rules**
   1. Expect movement toward common rule formats across departments.
   2. Could evolve into a federal standard or Treasury Board Secretariat directive in coming years.

### **🧩 Implications for ESDC**

* **Cost Reduction**: Faster, more consistent translation of policy to operations.
* **Better AI Assistants**: RaC as a backend for trustworthy chatbots and assistants.
* **Decision Support**: Scenario analysis, simulations, and legal guidance made accessible to non-lawyers.

Would you like a visual diagram showing how symbolic AI, generative AI, and hybrid approaches work together in this RaC framework?

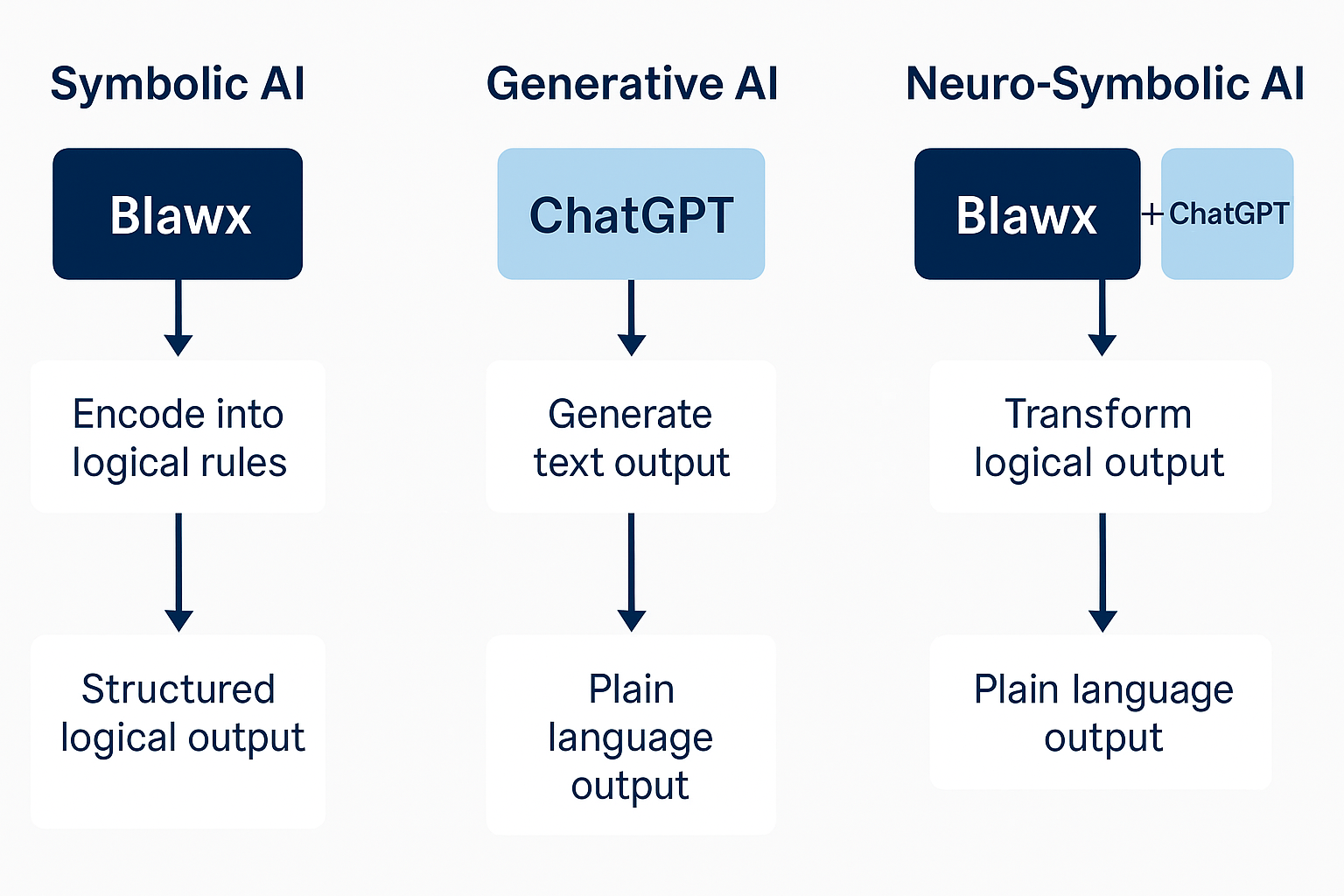
4o

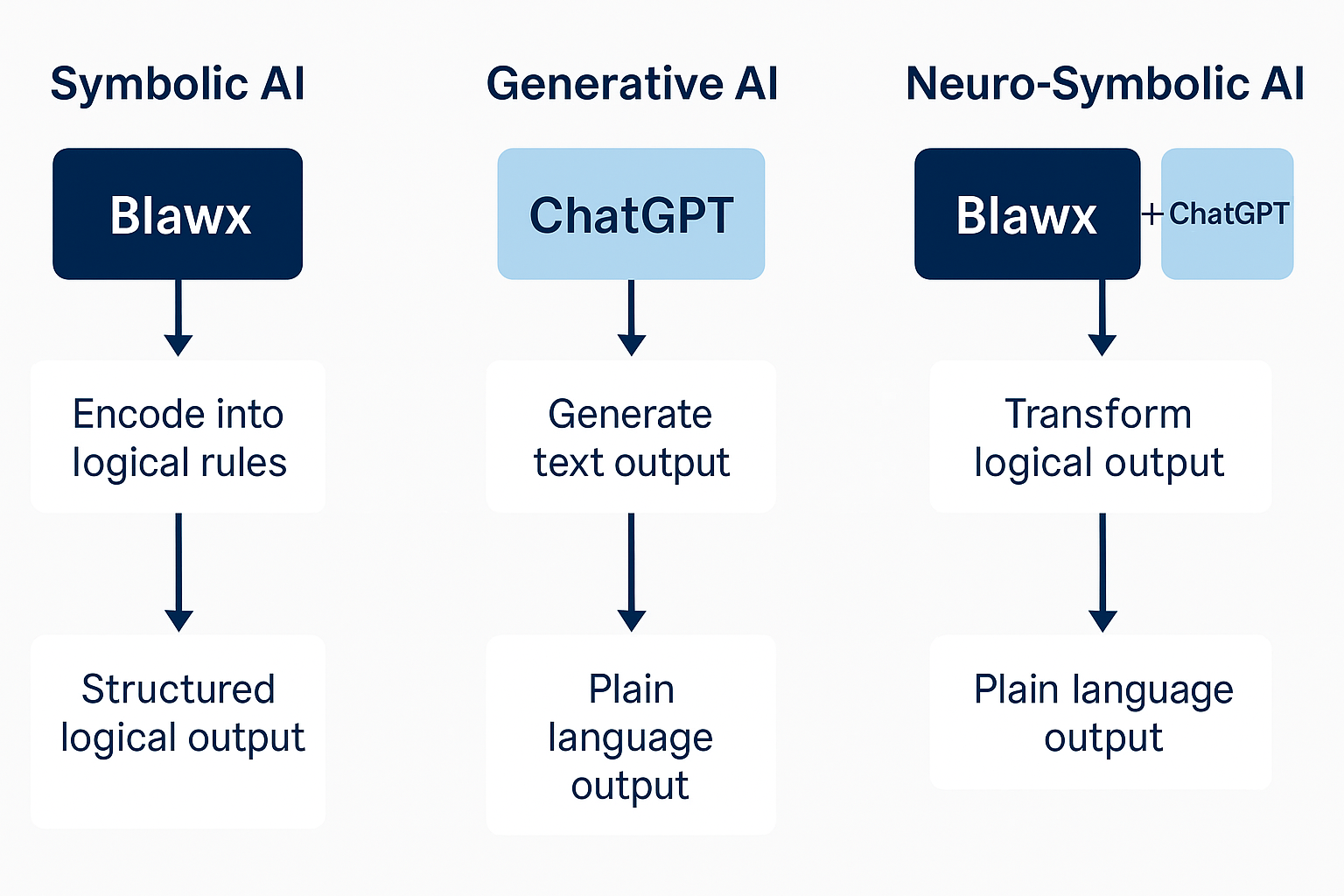
##### **You said:**

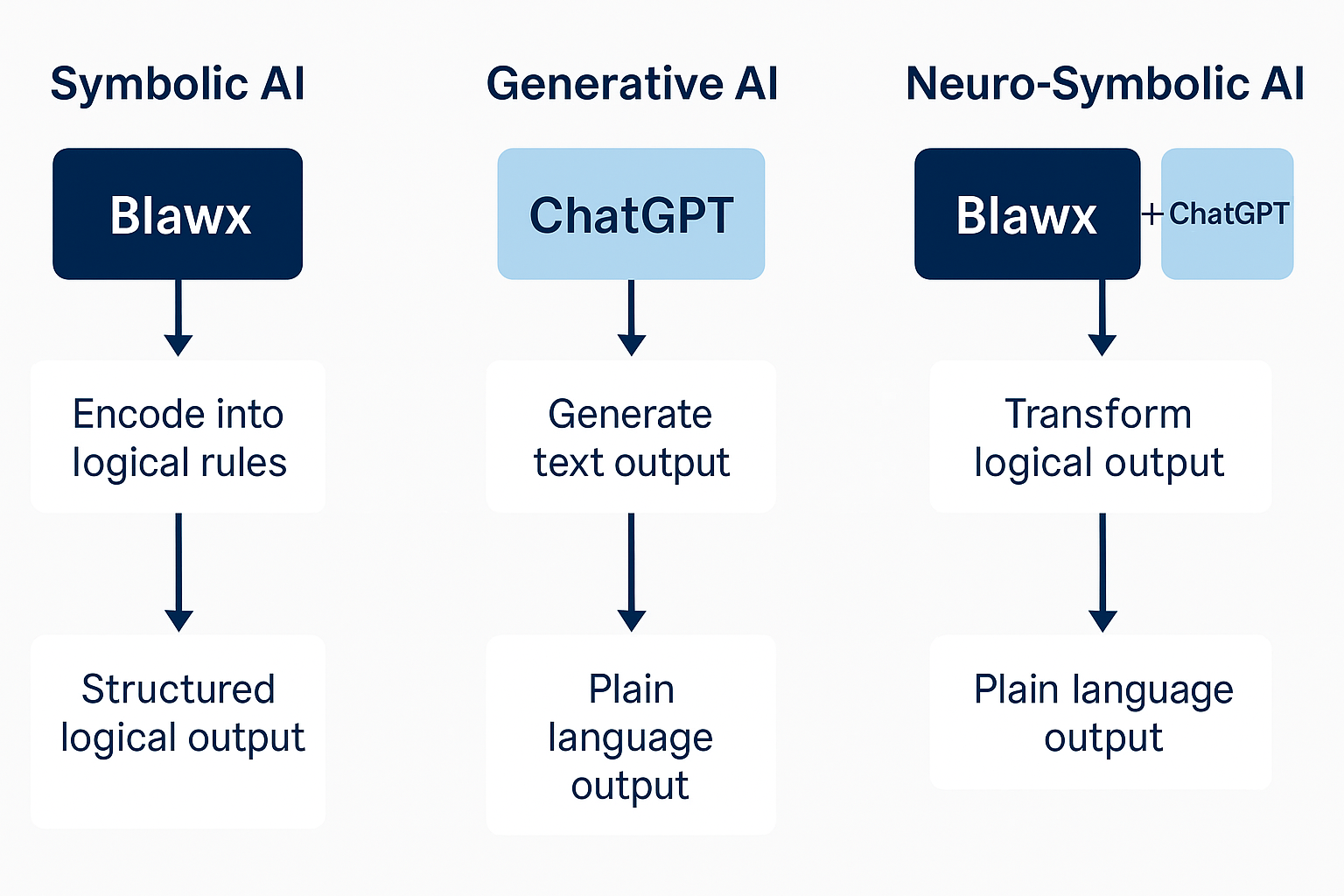
can you show me an example

###### **ChatGPT said:**

Image created







Rate this image

##### **You said:**

can you show me an example of how RaC would be used

###### **ChatGPT said:**

Sure. Let me walk you through a practical **Rules as Code (RaC)** example in the context of ESDC — specifically, for a **benefit eligibility rule** like the *Canada Child Benefit (CCB)*. This example will highlight how RaC transforms policy into machine-executable logic that can be reused across systems (e.g., eligibility checkers, chatbots, decision automation).

### **⚖️ Original Policy Rule (Human-Readable)**

"To be eligible for the Canada Child Benefit, the applicant must:

1. Be a resident of Canada for tax purposes;
2. Live with a child under the age of 18;
3. Be primarily responsible for the care and upbringing of the child."

### **🧠 RaC Version (Logic-Based)**

Using Blawx or another declarative logic language like Prolog or s(CASP), the rules might look like this:

prolog

CopyEdit

eligible(Applicant) :-  
 resident(Applicant, canada),  
 lives\_with(Applicant, Child),  
 age(Child, Age), Age < 18,  
 primary\_responsibility(Applicant, Child).

### **🛠️ How It's Used in Practice**

#### **1. Eligibility Calculator**

A service portal (like MyESDC) could use the logic to create a simple eligibility checker:

* **Input**: "I live in Quebec. My son is 5. I’m his primary caregiver."
* **Backend**: RaC engine evaluates the encoded rules.
* **Output**: "Yes, based on your input, you appear eligible for the CCB."

#### **2. Chatbot Integration**

EVA Chat or another AI chatbot uses the RaC backend:

* **User**: “Do I qualify for the child benefit if my niece lives with me?”
* **Chatbot**: Gathers required facts through questions, then queries the RaC logic.
* **Answer**: "If you are the primary caregiver and reside in Canada, you may be eligible."

#### **3. Simulation and Policy Design**

Policy analysts simulate the impact of changing the age threshold from 18 to 21:

* Just change one parameter: Age < 21
* Run hypothetical scenarios to assess eligibility impact across population segments.

### **✅ Benefits of Using RaC**

* **Machine-readable and human-readable** rules
* **Traceability**: Every decision can be explained through logical steps
* **Consistency**: Same logic used across apps, portals, and chatbots
* **Easy Updates**: Policy changes are reflected by changing a rule, not rewriting entire systems

Would you like me to show this same example as a visual flow or in a chatbot Q&A scenario?

##### **You said:**

Which RaC features EVA Chat and EVA Domain Assistant would need

###### **ChatGPT said:**

For EVA Chat and EVA Domain Assistant to support **Rules as Code (RaC)** capabilities effectively, they would need features that support **rule ingestion, interpretation, reasoning, and explanation** — while remaining accessible to non-programmers. Here’s a breakdown of the required RaC features per tool:

## **🧠 EVA Chat – RaC Features**

**EVA Chat is conversational** and suited to dynamic Q&A, decision support, and guidance. To enable RaC:

### **1. RaC-Aware Prompting Layer**

* Accept structured rules (e.g., Blawx-generated outputs) as context.
* Inject symbolic outputs as system prompts or tool-calls to guide the conversation.

### **2. Fact Collection via Dialogue**

* Ask follow-up questions to gather missing rule inputs ("Are you primarily responsible for the child’s care?").
* Store responses as structured variables (like applicant\_country = Canada).

### **3. RaC Execution API**

* Send collected facts to a reasoning engine (e.g., Blawx or Logic-LM).
* Receive a yes/no answer *with explanation steps*.

### **4. Transparent Reasoning Responses**

* Output structured, traceable reasoning: “Because you live in Canada and care for a child under 18, you qualify under paragraphs A and B.”

## **📚 EVA Domain Assistant – RaC Features**

**Domain Assistant is document-oriented and retrieval-based** — think policy ingestion, trusted content, structured metadata, and RAG (Retrieval-Augmented Generation).

### **1. RaC Rule Ingestion**

* Ability to ingest .blawx files or s(CASP)/Prolog rules as trusted content types.
* Treat RaC rules like structured data sources.

### **2. Metadata Tagging for Legal Concepts**

* Label documents or snippets with RaC categories (e.g., navigable\_water, primary\_caregiver).
* Enable context-aware retrieval tied to logical rules.

### **3. RaC Output Explanation Rendering**

* When an answer is sourced from logic (not just retrieval), provide rule-chain outputs: “Derived from CNWA section 2(a), confirmed by RaC logic.”

### **4. Cross-linking Rules and Regulations**

* For each encoded rule, link to its legal source.
* Allow users to inspect encoded rules in plain language and visual blocks.

## **🧩 Optional Advanced Features for Both**

* **Hybrid Reasoning Layer**: Combine LLM (ChatGPT) summaries with logic-based outputs from Blawx.
* **Rule Change Simulation**: Allow policy owners to test rule variations and see changes reflected in real-time answers.
* **Multilingual Rule Support**: Encode rules in both English and French to match Canadian bilingual legislative standards.

Would you like me to mock up a UI prototype for EVA Chat or EVA DA using one of these features, like rule validation or rule-based question flow?

##### **You said:**

is chatgpt capable of ingesting blax files

###### **ChatGPT said:**

ChatGPT itself **cannot directly ingest or execute Blawx files (.blawx)** or any other **s(CASP)/Prolog-based logic code** in a native way. However, it **can help you interpret, understand, or explain** the contents of such files **as text**, and assist in tasks like:

### **✅ What ChatGPT *Can* Do with Blawx or RaC Files**

1. **Explain Logic**: If you paste Blawx rules (in visual or text form), it can explain what they do.
2. **Suggest Improvements**: It can help you refactor rules or make them more readable.
3. **Translate Legal Text → Logic**: It can assist in drafting logic based on legal statements.
4. **Generate Plain Language Summaries**: Useful for hybrid RaC + GenAI approaches.

### **❌ What ChatGPT *Cannot* Do**

1. **Execute Blawx/s(CASP) logic**: It cannot run the rules and produce outputs (like true/false).
2. **Visualize Blockly Blocks**: ChatGPT can’t render or interpret the block-based UI.
3. **Trace Rule Reasoning**: It can’t simulate internal RaC inference like Blawx’s step-by-step logic chain.
4. **Verify Legal Accuracy**: It cannot validate if the encoding properly reflects statutory interpretation.

### **🧠 Workaround: Hybrid Architecture**

To make this work in EVA or any real use case:

1. **Blawx or another logic engine** does the reasoning.
2. **EVA (or ChatGPT)** receives:
   1. A natural language summary of the conclusion,
   2. The reasoning steps (from Blawx),
   3. Optionally, the original rule text.
3. ChatGPT generates the explanation or helps walk the user through the logic conversationally.

### **Example Flow**

plaintext

CopyEdit

User: "Am I eligible for the benefit?"  
ChatGPT (with Blawx backend):   
"Based on your answers, you meet the conditions in Rule 2.a.   
You live in Canada, have a child under 18, and are their primary caregiver.  
Therefore, the system concludes you're eligible."

If you're thinking of building a true EVA Chat + RaC integration, you’d need:

* A Blawx backend/API running logic,
* A conversational layer (ChatGPT) that parses outputs,
* A bridge that injects facts and collects results.

Would you like a reference architecture diagram or example API workflow for that?

##### **You said:**

before we go there, what are the benefits for users and business if EVA Foundation (Chat + DA) supported RaC?

###### **ChatGPT said:**

Supporting **Rules as Code (RaC)** in the **EVA Foundation** (Chat + Domain Assistant) could unlock **major benefits** for both **end users** and the **business**, especially in a policy-heavy environment like ESDC. Here's a structured view of those benefits:

## **👥 For Users (Employees, Citizens, Case Workers)**

### **1. ✅ Clear and Consistent Answers**

* RaC enforces rule logic directly from law or policy.
* Users get **uniform answers**—no variation across chat, portal, or forms.

*Example: Whether someone qualifies for EI is answered identically by the chatbot, web form, or internal eligibility checker.*

### **2. 💬 Explainable Decisions**

* RaC logic can show *why* a conclusion was reached.
* Users can understand the rule path (“Because you are under 18 and reside in Canada, you qualify…”).

Improves **trust** and reduces **appeals and confusion**.

### **3. 🕵️ Faster Eligibility/Assessment Results**

* Logic runs instantly.
* Instead of navigating a guide or policy manual, users just answer questions—*the system reasons for them*.

Think of a **self-serve chatbot or internal assistant** that gives eligibility outcomes in seconds.

### **4. 🌐 Language and Accessibility Friendly**

* RaC-based reasoning outputs can be translated into **plain language**, **multiple languages**, or **spoken-word interfaces**.
* No legalese; just facts and logic presented clearly.

## **🧭 For the Business (Policy, IT, Program Delivery, Legal)**

### **1. 🧱 Single Source of Truth for Rules**

* Write the logic once (e.g., Blawx or s(CASP)).
* Reuse it across chatbot, web portal, intake systems, simulations, etc.

Avoids duplication and **coding inconsistencies** across channels.

### **2. ⚖️ Legally Traceable Compliance**

* All decisions are explainable and tied to specific policy provisions (including sections and clauses).
* Supports audit, transparency, and **accountability under the rule of law**.

Essential for programs like EI, CPP, OAS, CCB, or any compliance-heavy benefit.

### **3. 🔄 Faster Policy-to-Code Cycle**

* Policy changes (e.g., new age cut-off, exception rule) only require updating logic once in the RaC model.
* No more waiting for developers to rewrite logic in 5+ systems.

Cuts time-to-implement **from months to days or hours**.

### **4. 📉 Reduced Errors and Litigation Risk**

* Errors caused by misinterpreting policy or inconsistent logic are minimized.
* Reduces **rework, complaints, and legal challenges**.

### **5. 📊 Supports Simulation and Impact Analysis**

* Analysts can run “what if” simulations to understand how rule changes affect populations.
* Helps **evidence-based policy design**.

### **6. 🛠️ Modular Integration into EVA DA & Chat**

* Rules can live as assets in EVA DA projects.
* Chatbots (EVA Chat) dynamically query the rules based on user input.

## **🔄 Summary Table**

|  |  |
| --- | --- |
| **Stakeholder** | **Benefit** |
| **Citizens/Users** | Fast, fair, and transparent answers |
| **Case Workers** | Confidence that rules are applied correctly |
| **Policy Teams** | Easier to prototype, test, and deploy new rules |
| **Legal/Compliance** | Traceable logic supports legal defensibility and rule of law |
| **IT/Developers** | Decouples rules from hard-coded logic, reducing dev overhead |
| **Executives** | Faster, cheaper, safer delivery of services |

Would you like a one-pager or executive slide summarizing these benefits for sharing with stakeholders?

4o

##### **You said:**

what is the difference between RaC and standard if then else programming

###### **ChatGPT said:**

Great question — and an important one when advocating for **Rules as Code (RaC)** in a government or enterprise context.

## **🧠 Rules as Code vs. Traditional "If-Then-Else" Programming**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Rules as Code (RaC)** | **Standard If-Then-Else Programming** |
| **Purpose** | Encode legal/policy rules in a machine-executable, traceable form | Implement business logic via imperative code |
| **Approach** | **Declarative** – Describe *what* is true, not *how* to compute it | **Imperative** – Specify *how* to compute step-by-step |
| **Who Writes It** | Often written or reviewed by policy experts (e.g., using Blawx) | Written by developers in programming languages |
| **Explainability** | High – can answer “why” or “why not” a conclusion was reached | Low – requires code tracing or logging |
| **Rule Transparency** | Rules are explicitly stated and modular | Rules are embedded deep in code logic |
| **Adaptability** | Easy to change/update rules without breaking systems | Changing logic requires dev, testing, regression |
| **Traceability to Law** | Can be directly linked to policy sections or legislation | Often abstracted away or reinterpreted |
| **Reusability Across Systems** | High – one source of truth used across apps, chatbots, and workflows | Low – logic copied/re-implemented across systems |
| **Formal Reasoning Support** | Yes – uses logic engines (e.g., s(CASP), Prolog) to infer conclusions | No – runs step-by-step without formal reasoning |

## **🧪 Example Comparison**

### **✏️ RaC Version (Logic-Based – Declarative)**

prolog

CopyEdit

eligible(Applicant) :-  
 age(Applicant, Age), Age >= 65,  
 residency(Applicant, canada),  
 contribution\_years(Applicant, Years), Years >= 10.

* Clearly declares *what* conditions must be true for someone to be eligible.
* Can answer **why** someone *is* or *is not* eligible.

### **🧱 If-Then-Else Version (Imperative – Procedural)**

python

CopyEdit

def is\_eligible(applicant):  
 if applicant.age >= 65:  
 if applicant.residency == "canada":  
 if applicant.contribution\_years >= 10:  
 return True  
 return False

* Focuses on *how* to process the rules procedurally.
* Harder to trace to legal text or debug why a user was deemed ineligible.

## **🧭 Why It Matters for Government and ESDC**

|  |  |
| --- | --- |
| **Goal** | **Why RaC Helps** |
| Consistent rule application | Everyone uses the same logic source |
| Transparent decisions | "Why am I not eligible?" → clear, traceable answers |
| Agile policy implementation | Rule change = rule update, not full system recoding |
| Trust and fairness | Reduces black-box decisions, improves confidence |
| Automation & AI integration | RaC rules can power chatbots, calculators, simulations |

In short, **RaC is purpose-built to handle complex, evolving, explainable policy logic** in ways that procedural code can’t — or at least, not cleanly or safely.

Would you like this explanation in a one-slide format for presentation purposes?

4o

##### **You said:**

what does that mean exactly? 6. 🛠️ Modular Integration into EVA DA & Chat Rules can live as assets in EVA DA projects.

###### **ChatGPT said:**

Great follow-up. Let's unpack what this means in concrete, operational terms:

## **🔧 What Does It Mean to Say:**

**“Rules can live as assets in EVA DA projects”?**

In EVA Domain Assistant (DA), each application or project space contains **curated data assets** like documents, files, prompts, and metadata. By saying **“rules can live as assets”**, we mean:

### **✅ 1. Rules are Stored Like Documents**

* Just like you upload PDF guides or policy manuals into EVA DA, you could **upload a .blawx file** or any structured **machine-readable rule set** (e.g., Prolog, YAML, JSON logic).
* These become part of the **knowledge base** for that project.

### **✅ 2. Rules Can Be Tagged, Indexed, and Versioned**

* Each rule file can have:
  + **Metadata** (e.g., version, source law, effective date)
  + **Tags** for retrieval and filtering
  + **RBAC controls** to restrict visibility/edit rights
* This allows knowledge owners (e.g., policy teams) to maintain and update rules without coding.

### **✅ 3. Rules Can Be Queried Programmatically**

* When a user asks a question in EVA DA or EVA Chat:
  + The system **retrieves related documents**
  + AND/OR **executes a logic query** if the question involves a rules-based outcome (e.g., eligibility, compliance)
* The rule file acts like an **active data source** — not just a passive document.

### **✅ 4. Shared Logic Across Tools**

* The same rules stored in EVA DA can be **called by EVA Chat** (via API or function call).
* This means one RaC file powers:
  + Chatbot explanations
  + Calculators or checkers
  + Decision logic in workflows

### **✅ 5. Transparent Governance**

* Because rules are treated as **named, stored, and visible assets**, they support:
  + Change tracking (who updated what, when)
  + Auditing and traceability
  + Policy simulations or testing (try new rule sets without impacting production)

### **💡Analogy:**

Think of EVA DA like SharePoint for smart rules — you don’t just store PDFs, you store logic that systems can *use*, *explain*, and *trust*.

Would you like a short mock-up showing what a RaC rule asset entry could look like inside an EVA DA project UI (title, metadata, rule viewer, etc.)?