ChatGPT

Plan for Regenerating ACA v4 Module Documentation

Overview and Objectives

In this task, we will **regenerate every ACA v4 module document** (both the standard version and the detailed version, where applicable) as clean, up-to-date PDFs. The goal is to produce a **complete, consistent set of module documents** that adhere to the latest architecture rules and module specifications. Each regenerated module document must strictly align with the **ACA v4 Master Rulebook** (which defines the overall architecture, layering, execution order, safety rules, and module responsibilities) and the **detailed spec for that module** (covering its inputs/outputs, internal logic, pseudocode, and failure modes). We will eliminate outdated or duplicated content from previous drafts, resolve any discrepancies between "standard" and "detailed" versions, and apply a uniform formatting style consistent with ACA cognitive architecture documentation norms. Finally, we will compile the finished PDFs into a single archive (ZIP) for easy distribution.

Key objectives:
- **Accuracy:** Ensure each module's documentation reflects the **validated system logic** (the real behavior of the current ACA v4 system).
- **Consistency:** Maintain **cross-module consistency** so that outputs of one module match the expected inputs of the next (for example, the **Process Engine's** reasoning plan must cleanly feed into the **Decision Tree Builder** and into **AIM Phase 1**, exactly as the system is designed to do).
- **Clarity & Safety:** Incorporate all relevant rules from the Master Rulebook regarding how modules interact, layer by layer, and enforce safety or policy checks. The content should be clear, readable, and traceable (each requirement or rule should be easy to locate in the docs).
- **Complete Specs:** Cover every aspect of each module's spec: **purpose/responsibility**, **inputs & outputs**, **process flow (with pseudocode)**, and **failure modes/error handling**.
- **Formatting:** Present the information in a clean, professional format (with proper section numbering, headings, and formatting) that aligns with ACA documentation standards for cognitive architectures.

Below is a structured plan outlining how to achieve these objectives step by step:

Step 1: Gather Current Reference Materials

Begin by collecting all the **source materials** that will guide the regeneration of the documents:

- **ACA v4 Master Rulebook:** This is the authoritative guide on the architecture. It defines how the system is structured (module layers, execution order), how modules should behave (responsibilities, interfaces), and how safety and lawfulness are enforced across the system. We will use this to check that each module doc conforms to global requirements (e.g. no module violates execution order or bypasses safety enforcement rules).

- **Module Specifications:** For each module in ACA v4, locate its detailed specification. This could be in the form of design documents, previous documentation drafts, or internal wiki pages. The spec should include:
- **Module purpose and responsibilities** – what is this module supposed to do in the overall architecture?
- **Inputs and Outputs** – what data does it receive, and what does it produce? (Including data formats or structures expected.)
- **Internal logic and processing steps** – the algorithm or decision process it follows, often provided as pseudocode or flowcharts.
- **Safety or policy checks** – any compliance with global rules or filters the module must apply (for example, checking user inputs against disallowed content or ensuring responses comply with safety guidelines).
- **Failure modes** – how the module handles errors, timeouts, or unexpected inputs (does it retry, fail gracefully and return a specific output, trigger a recovery module, etc.).
- **Previous Module Documents (v4 drafts):** If there are existing documents for modules (both "standard" and "detailed" versions from earlier drafts), gather them for reference. These will be combed for any useful content but **we will not copy them verbatim** – instead, we'll use them to ensure we don't miss any important point from the specs. We also need to identify any outdated or duplicated content in these drafts so we can remove or update it.

Outcome of Step 1: We have a complete set of references – the Master Rulebook and each module's spec – ready for review. We understand the expected correct behavior of each module and the rules that apply across the architecture.

Step 2: Identify All ACA v4 Modules and Document Versions

Next, make a **comprehensive list of all modules** in the ACA v4 architecture. According to the architecture, ACA v4 is composed of many interacting modules (Ghoststack's cognitive substrate is described as having dozens of modules working in concert [1], ensuring lawful, stable cognition). We should include every module that is part of the v4 system, for example (not an exhaustive list, just illustrative):

- **Process Engine** – responsible for generating the internal reasoning plan or chain-of-thought for a query.
- **Decision Tree Builder** – uses the reasoning plan to form a structured decision tree of tasks or questions to answer.
- **AIM Phase 1** (and any subsequent AIM phases) – the "Action/Answer Integration Module" (for instance) that takes the decision tree or reasoning outputs and begins to formulate the final answer or action in phases.
- **Context Manager** – maintains conversation or session context if applicable.
- **Knowledge Retrieval Module** – fetches information from a knowledge base or memory as needed.
- **Safety/Compliance Module** – checks for policy violations and ensures responses will be safe.
- **Execution Orchestrator** – controls the pipeline sequence (ensuring the modules execute in the correct order per the rulebook's defined layering).
- **Memory/State Module** – stores any persistent state or long-term memory needed between operations.
  (Note: Replace or extend this list with the actual modules in ACA v4; the above are hypothetical examples.)

For each identified module, note whether there are two versions of documentation required: - A **Standard Module Document** – typically a concise spec intended maybe for general reference. - A **Detailed Module Document** – a more in-depth version with expanded explanations, rationale, and possibly more examples or deeper pseudocode.

If both exist or are needed, we will regenerate both, making sure they are consistent with each other (differing only in level of detail, not in factual content).

Outcome of Step 2: A checklist of all modules and the documents we need to produce for each. For instance: "Process Engine (standard + detailed), Decision Tree Builder (standard + detailed), AIM Phase 1 (standard + detailed), [Module X] (standard only), [Module Y] (standard + detailed)", and so on.

Step 3: Review and Align with the Master Rulebook

Before editing individual module docs, thoroughly review the **ACA v4 Master Rulebook** for any guidelines that affect documentation content. Key areas to pay attention to include:

- **Architecture Layering:** The rulebook likely defines layers or tiers of modules (for example, input processing layer, reasoning layer, output generation layer, safety layer, etc.). Each module belongs to a layer and may only receive input from certain layers and output to certain layers. Ensure the documentation of each module clearly states its layer and adheres to those boundaries. (For example, if the rulebook says that *modules in the reasoning layer cannot directly access the user interface*, then the docs should reflect that – i.e., the Process Engine does not directly produce user-visible output, it passes data to output modules.)
- **Execution Order:** The sequence in which modules run. Some architectures enforce a strict order (e.g. first the Process Engine runs, then the Decision Tree Builder, then AIM Phase 1, etc.). The Master Rulebook will specify this pipeline or any parallel execution rules. We must verify that each module's doc is consistent with this order (e.g. it might have a section "Upstream Dependencies" and "Downstream Hand-offs" describing what comes before and after it in execution). Update any module descriptions that have outdated ordering assumptions.
- **Safety Enforcement:** Critically, the rulebook outlines how **AI safety and policy enforcement** is integrated. Some modules might have explicit responsibilities for safety (like a content filter module or a rule-checker), and others may need to call these or adhere to their decisions. For documentation:
- Emphasize any points where a module must perform a safety check or obey a policy (for instance, "Before generating the reasoning plan, the Process Engine calls the Safety module to ensure no disallowed content is in the input query.").
- If the Master Rulebook requires that *every* module double-check outputs against certain constraints, ensure each module's pseudocode includes that step.
- Highlight which module acts as the final gatekeeper for responses (if any) and make sure its document reflects the rulebook's safety protocol precisely.
- **Module Responsibilities:** The Master Rulebook might enumerate the core responsibility of each module or each layer. Cross-check that against our module specs. If a previous draft had mis-stated a module's role (e.g. maybe an earlier doc had the Decision Tree Builder also doing some answer formatting, but the rulebook says that should happen in AIM Phase 2), correct it now. The documentation should clearly state the **authorized role** of the module and nothing beyond that (to prevent overlap or confusion between modules).

- **Inter-module Interfaces:** The rulebook could define standard data interfaces or formats for data passed between modules (like a specific data structure for the reasoning plan, or a standardized intermediate representation of the user's query). All module docs should use these standard terms and formats. For example, if the reasoning plan is formally called a "Solution Graph" in the rulebook, then the Process Engine doc should say it outputs a "Solution Graph" (not a "list of thoughts", if that was an old term). Consistency in terminology here is crucial for traceability.

As we review these points, **mark any discrepancies** between the rulebook and the current drafts of module docs. These are the things we will fix in the rewriting process. In some cases, it might require adding new sections or clarifications in the module docs (if the rulebook introduced new rules or architecture changes in v4 that weren't reflected in older docs).

Outcome of Step 3: A clear list of rulebook alignment requirements for each module. For example: "Update Module X to mention it runs after Module W and before Module Y; Add safety check step to Module Z's pseudocode; Change terminology 'plan' to 'Reasoning Graph' across modules; Remove any mention of Module A calling external API, now forbidden by rulebook," etc.

Step 4: Update Module Content (Inputs, Outputs, Logic, Failures)

Now we proceed module by module, **rewriting and updating the content** according to the latest specs and the alignment needs identified:

For each **ACA v4 module document**, ensure the following sections are present and correct:

- **Module Purpose & Responsibilities:** Start the document with a brief description of what the module does and **why** it exists in the architecture. This should be one to two paragraphs giving context. Make sure this aligns with the Master Rulebook's definition of the module's role. (For a detailed version of the doc, this section might be longer and include rationale or examples; for a standard version, it might be a succinct summary.)
- **Inputs:** Clearly list what inputs the module receives. Include:
- The source of each input (e.g. "Receives the `Reasoning Plan` object from the Process Engine" or "Accepts natural language query from the user via the Interface Module").
- The format or structure of the input (data types, any schema, or an example if helpful).
- Any prerequisites (e.g. "Requires that the Context Manager has already populated the context object").
- **Outputs:** Clearly list the outputs the module produces and where they go next:
- What data is output, in what format.
- Which module or system component will consume that output.
- If the module has multiple outputs (e.g., a primary result vs. a log or a feedback signal), document all of them.
- **Process Flow / Internal Logic:** This is typically given as **pseudocode or a stepwise algorithm description**. We must update this pseudocode to reflect the *correct current logic*:
- Follow the **pseudocode structure** defined in the module spec (for example, if the spec uses a particular style – such as numbered steps or indentation conventions – adhere to that).
- Include all major steps: reading inputs, any decision branches, loops, calls to other modules or utilities, and producing outputs.

- **Incorporate safety and rule checks** explicitly. For instance, if at Step 3 the module must call a safety filter, include that as a step in the pseudocode (with a note if needed, e.g., "if safety_check fails, abort process and return error to [Recovery Module]").
- Ensure the pseudocode uses the updated terminology and interfaces from the rulebook (as noted in Step 3).
- Ensure that any error conditions or boundary cases are handled in the pseudocode. For example, "if input reasoning plan is empty, then log an error and return an empty Decision Tree to avoid crash."
- **Failure Modes & Error Handling:** After the main flow, have a subsection detailing how the module behaves when things go wrong. This includes:
- Possible failure conditions (e.g. "timed out waiting for response from knowledge base", "received ill-formed data from previous module", "internal exception during processing").
- For each, describe what the module does. Does it retry? Does it propagate an error signal to a higher-level controller? Does it default to a safe output?
- Include any **fail-safe mechanisms** mandated by the architecture. The Master Rulebook might specify certain modules must never fail openly; instead they must either attempt a recovery or hand off to a designated Recovery module. Make sure to describe those behaviors.
- If the module has a "detailed" doc version, you might include more analysis of failure modes (like why certain failures occur and how they were mitigated in design). In the standard doc, just listing them and responses is enough.
- **Example (if applicable):** Some detailed docs might include a short example scenario illustrating the module's function. If the spec or previous drafts have an example, check that it's still valid under v4 logic. Update any part of the example that is outdated. (For instance, if an old example showed a user query being parsed by this module but in v4 maybe parsing moved to a different module, update the example or choose a new one that correctly shows this module's function.)
- **Interface Details (if needed):** If the module interfaces are complex, we might include a small section on interface specification (like data schemas or API call format). Ensure these details match the current implementation.

We will rewrite each module document according to the above, **removing any content that is duplicated or no longer accurate**. For example, if previous drafts repeated the description of a common data structure in every module doc, we might trim that down and instead reference a central definition (unless each doc must stand alone – in which case ensure the definition is consistent everywhere). Outdated content (e.g., references to "v3" behavior or deprecated features) should be deleted to avoid confusion.

Throughout the rewrite process, **maintain consistency** in style: - Use the same terms for the same concept across all modules. (If one doc says "Reasoning Plan" and another says "Thought List" to mean the same thing, choose one term per the rulebook and apply it everywhere.)
- Use a clear and formal tone appropriate for technical documentation.
- Keep paragraphs short and focused. Where it makes sense, use bullet points or numbered lists (for instance, listing inputs, outputs, or failure cases as bullet points can improve readability).

Outcome of Step 4: Each module now has a rewritten content covering purpose, I/O, logic (with pseudocode), and failure modes, all aligned with the latest spec and rulebook. The content is internally consistent and free of irrelevant or repeated information.

Step 5: Ensure Cross-Module Consistency

With drafts of all module docs updated individually, we now **cross-review them for consistency with each other**. This involves checking the touchpoints between modules:

- **Data flow consistency:** Verify that if Module A's document says it outputs X in a certain format, then Module B (which consumes X) documents that same format for input. For example, the **Process Engine** outputs a "reasoning plan" and the **Decision Tree Builder** input section should explicitly accept that "reasoning plan" object. The two docs should not conflict on what that object contains. If any mismatch is found (e.g., one doc had outdated info), fix it now in both places.
- **Sequential logic checks:** Make sure that the described **execution order** is consistent. If the Orchestrator triggers modules in a certain sequence, each module doc should implicitly or explicitly reflect that (like by mentioning "prerequisite: Module Y must have completed" or "sends result to Module Z"). All modules should agree on the order and dependencies.
- **No gaps or overlaps:** Ensure that collectively, the documents don't describe any missing step or double-work:
- If one module says "I leave task Q for the next module," confirm that the next module indeed covers task Q.
- Conversely, if two modules both claim responsibility for the same function, that's an overlap to resolve according to the official spec. Adjust the responsibilities so each function is clearly owned by one module as per design.
- **Consistent safety handling across modules:** Since safety enforcement is a theme, verify that at least one module (or multiple at different layers) are clearly covering it, and that no module contradicts another in approach. For example, if both Module M and Module N perform a profanity filter on output, is that intentional (defense in depth)? If yes, document it consistently; if not, perhaps only one should do it – clarify per rulebook and update the other to remove mention of it.
- **Version synchronization (standard vs detailed):** If a module has two versions of documentation, compare them side by side. The **Standard** version should be a condensed summary of the **Detailed** version's content, **without any contradictions**. Ensure that:
- All facts (inputs, outputs, etc.) match exactly.
- The order of steps and logic is the same (the detailed might just break it down further or explain rationale).
- If the detailed version added any new info (like an additional failure case or an insight), verify if the standard version should include at least a mention of it (or if it's okay that it's omitted for brevity). Nothing critical should exist *only* in the detailed version without at least being correct in the standard version.
- Remove any leftover content that was present in one version but was outdated in the other. Both need to be updated to v4 spec.
- **Traceability:** One aim is *traceable documentation*, meaning a reader or auditor can trace requirements from the rulebook into the module docs. While not a direct content cross-check, consider adding cross-references (if applicable) or at least consistent section numbering that makes it easy to refer back and forth. For example, if the Master Rulebook had rules numbered, sometimes module docs might cite "(per Rule 3.2 of Master Rulebook, this module must do X)". If that's a practice in ACA docs, ensure those references are correct and updated.

At this stage, it can help to do a quick **team review or peer review**: have another engineer or team member skim through the set of module documents to catch any inconsistencies we might have missed.

Sometimes, explaining the flow from one module to the next out loud or via a simple diagram can reveal if the documentation has any logical gaps. If any are found, update the relevant docs for consistency.

Outcome of Step 5: The full collection of module documents reads as a **cohesive, integrated description of the ACA v4 system**. All hand-offs between modules are clearly documented and match on both sides. Standard and detailed versions (where both exist) are aligned with each other. Terminology and style are uniform across the documents.

Step 6: Apply ACA Formatting and Style Norms

Now that content is finalized, ensure that each document follows the **ACA cognitive architecture documentation format guidelines**. This typically includes:

- **Section Numbering:** Use a structured numbering scheme for sections and subsections (e.g., 1.0, 1.1, 1.2, 2.0, etc.). Often, a standard module document might have major sections like: 1. Introduction/Purpose, 2. Inputs, 3. Processing Logic, 4. Outputs, 5. Failure Modes, etc. The detailed version might intersperse additional subsections (like 3.1 Algorithm Steps, 3.2 Pseudocode, etc.). Follow whatever numbering convention is used in ACA docs so that it's easy to reference sections across documents.
- **Headings and Formatting:** Ensure all headings are consistent in style (e.g., all second-level headings might be bold or a certain font size in the PDF template). Use formatting to improve clarity:
- Bold or italicize key terms on first use if needed (for example, **Reasoning Plan** could be in bold when defined).
- Use monospaced font for code or pseudocode blocks. Pseudocode should be indented properly and maybe within a markdown-style code block or appropriate environment so it stands out clearly.
- Use bullet points or numbered lists for enumerations (like lists of inputs or failure cases) to avoid long paragraphs. The goal is **readability** – someone skimming should easily catch the key points.
- **Tables/Figures:** If the module spec includes any tables (perhaps comparing old vs new behavior, or listing rules) or figures (like a flowchart or architecture diagram snippet), ensure these are properly formatted and labeled. Update any figure captions to match the new content. If any diagrams were outdated, regenerate them if possible (though the task primarily focuses on text/PDF; if diagrams exist in source form and need updating, that might be a separate sub-task).
- **Traceability and labeling:** If the ACA norm requires tagging certain requirements or rules IDs, make sure we included them. For example, some docs include requirement IDs like [AR-5] to denote "Architecture Rule 5" from the Master Rulebook. If such references exist, verify each one points to a valid current rule. If not used, consider whether adding reference to key rules could improve traceability (but avoid cluttering the document unnecessarily).
- **Document Metadata:** Each PDF should have a clear title page or header indicating the module name, version (ACA v4), and whether it's the standard or detailed version. For example, the header might say "ACA v4 – Process Engine Module Specification (Detailed Version)" along with date and perhaps the author or team name. Update any version numbers or dates from the old drafts to the current ones.
- **Remove Redundancy:** Ensure that if some explanatory text was repeated in multiple places, we either consolidate it or reference it. A common example: if every module document had a copy-pasted paragraph explaining what ACA is, perhaps it's better to remove that from each and just keep module-specific info (assuming readers have the Master Rulebook or an overview doc for general info). The regenerated docs should be **lean yet complete** for their specific topic.

- **Proofreading:** Finally, do a thorough proofread on each document for grammar, spelling, and clarity. Since we edited a lot, ensure no placeholder text or editor's notes remain. Check that all cross-references (if any) are updated (e.g., if the detailed doc says "see Section 3.2 above" but numbering changed, fix that). The tone should be consistent—use either imperative (common in spec: "The module **shall** do X") or descriptive third-person ("This module does X"), whichever style the ACA docs typically use.
- **Compliance with Norms:** The ACA cognitive architecture norms might also include guidelines like "Do not use ambiguous terms", "Prefer shall/will for requirements statements", etc. Ensure our text follows these. For enforceable traceability, often requirements are stated as clear, testable statements. So if needed, reword any fuzzy descriptions into clear statements. For instance, instead of "the module tries to filter unsafe outputs", say "the module filters out any response content that violates safety policies before forwarding output (per Master Rulebook Rule 4.5)". Clarity is key.

Outcome of Step 6: Polished, well-structured documents that look professional and are easy to navigate. Each module PDF is formatted consistently, making the set feel like a unified documentation package. At this point, the content is finalized and the presentation is refined.

Step 7: Generate PDF Files for Each Module Document

With the final text and formatting in place (likely in a source format like Markdown, Word, or LaTeX depending on our workflow), the next step is to **export each document to PDF**:

- Ensure any styling from Step 6 is preserved in the PDF output. If using a tool or markup (like Markdown to PDF converter or LaTeX), double-check that all sections, indentation, bullet points, and code blocks appear correctly in the PDF. Pay special attention to page breaks (avoid orphaned headings at page bottom, etc., by adjusting spacing or using page-break commands if necessary).
- Check that diagrams or tables (if any) are not cut off and render with good resolution. Lower resolution images are acceptable if clarity is maintained; if any image is blurry or too low quality, see if it can be replaced or regenerated at a better quality.
- Populate PDF metadata (like Title, Author, etc.) if our PDF tool allows it, so that each file is identifiable. For example, Title could be "ACA v4 – [Module Name] Spec".
- Naming convention: Name each PDF file clearly and consistently, e.g., `ACA_v4_ModuleName.pdf` for standard and maybe `ACA_v4_ModuleName_Detailed.pdf` for detailed. This will help when we package them.
- After generation, do a quick scan of each PDF to ensure no content is accidentally missing or formatted incorrectly. Sometimes conversion can introduce weird spacing or miss fonts – make sure everything looks as intended. Also verify that all pages are present (compare page count if the source had it).

We should now have a folder with all the final PDFs, one per module (or two per module if detailed included).

Outcome of Step 7: High-quality PDF documents for each ACA v4 module, ready to be delivered. Each PDF stands on its own for its module, and together they cover the whole system.

Step 8: Package the Final PDFs into an Archive

As a last step, **create a ZIP archive** containing all the regenerated module PDFs for easy download and reference:

- Gather all the PDF files into a single directory. Double-check the filenames one more time for consistency and correctness (typos in module names, etc., should be corrected now).
- Create a ZIP file (e.g., `ACA_v4_Module_Docs.zip`). Ensure the zip structure is flat (all PDFs in the root of the archive) unless there's a reason to organize them (if the list is long, sometimes grouping by layer or subsystem could be useful, but given they all belong to v4, a flat list named by module is fine).
- The zip should also ideally include a **README or index** (maybe as a short text file or as part of the archive comment) listing the modules included. Alternatively, we could rely on clear filenames. If we want to be extra helpful, an index PDF or text could list all modules and their brief descriptions, but that might be beyond scope unless already available.
- Once zipped, test extracting it to ensure no issues. The final size should be noted if needed (but since these are text-heavy PDFs, it shouldn't be very large).
- This archive can now be provided to stakeholders as the **complete ACA v4 module documentation set**.

Outcome of Step 8: A single downloadable archive containing all module documentation PDFs, ready for distribution. This fulfills the requirement of packaging the final modules into a deliverable bundle.

Step 9: Validation and Final Review

(Although not explicitly requested, it's good practice to validate that what we regenerated meets all requirements.)

- **Check Against Master Rulebook:** Pick a few key rules from the Master Rulebook and verify that each is addressed in the relevant module docs. For example, if Rulebook says "All output modules must pass through the Safety Module before delivering content," ensure the output-generating module docs mention that.
- **Stakeholder Review:** If possible, have the architect or person responsible for the Master Rulebook review one module doc to ensure the alignment is satisfactory. Incorporate any feedback uniformly across all docs.
- **No Partial Draft Content:** Confirm once more that we did **not carry over any outdated partial drafts** text. Everything should either be newly written or thoroughly revised to be current. There should be no "TBD" sections or remnants of older version numbering that could confuse readers.
- **Detailed vs Standard parity:** Make sure again that no version mismatch slipped in. It might help to generate a quick diff between the standard and detailed texts (understanding differences are expected, but looking for unintended discrepancies).
- **Final Formatting Pass:** Ensure each PDF starts with a clear title and perhaps a brief introduction so that even if someone opens one out of context, they know what they're reading (including that it's ACA v4). Ensure page numbers, footers/headers if used, are correct.

After this, we can confidently say the ACA v4 module documents have been regenerated to a high standard.

Conclusion

By following the steps above, we will have a **complete, up-to-date, and consistent set of ACA v4 module documents**. These documents will strictly adhere to the ACA v4 Master Rulebook's architecture and rules, reflect each module's detailed specifications (with correct pseudocode and failure handling), and maintain cross-module consistency for seamless integration. The formatting and organization will make them easy to read and audit, supporting enforceable traceability throughout the cognitive architecture. Finally, delivering them as polished PDFs in a bundled archive will make distribution straightforward.

This regeneration effort ensures that anyone referencing ACA v4 documentation – whether developers, new team members, or external auditors – can rely on these PDFs as the single source of truth for how the cognitive architecture's modules function and interact. The system's design, from the high-level rules down to each module's pseudocode, will be clearly documented and aligned, paving the way for robust implementation and maintenance of ACA v4.

---

[1] #cuas #uas #defense #security #drone #innovation | James Acuna | 13 comments
https://www.linkedin.com/posts/james-acuna-1a07451b5_cuas-uas-defense-activity-7370448445008146432-2YQc