Declarative Multi-Agent Systems: From Using AI to Building AI

Introduction: From 'Using AI' to 'Building AI' – A Critical Leap

In late 2025, an open-source AI agent project called OpenClaw burst onto the scene, attracting millions of visitors and over 100,000 GitHub stars within a single week. 'Your assistant, your machine, your rules' – the slogan ignited enormous enthusiasm across the Global North's technology communities. Almost simultaneously, several technology giants rushed to release their own 'AI agent' products, each claiming to revolutionise the very nature of knowledge work. For technology professionals in the Global North, this may represent yet another exciting cycle of technical iteration; but for researchers, movement leaders, and intellectuals in the Global South, a more fundamental question demands attention: **when everyone is celebrating the technical breakthroughs of AI agents, very few are asking who owns and controls these tools, and whose agenda they serve.**

This is not a rhetorical question. When a researcher in the Global South uses the ChatGPT chat interface to assist her writing, she is a **consumer** of AI – consuming a product designed by others, working under rules set by others, on platforms owned by others, with data selected by others. She can, in principle, paste her methodology into the prompt and save the output as a file – but these are crude workarounds, not systemic capabilities. The system's behavioural logic remains opaque to her, the embedding of methodology is ad hoc and fragile, and knowledge accumulation depends entirely on manual effort outside the platform. Each conversation is an island; nothing compounds.

But if this researcher possesses a research system composed of dozens or even hundreds of agents – each with clearly defined responsibilities and quality standards, capable of automatically searching hundreds of web pages, processing hundreds of documents, analysing according to her chosen theoretical framework, and generating research reports with thousands of citations – then she is no longer a consumer but a **builder**. The system belongs to her, the methodology is embedded by her, the accumulated knowledge belongs to her institution, and the entire research process is transparent and under her control.

The leap from consumer to builder is a critical step for the Global South in reclaiming technological sovereignty.

The technical foundation upon which this leap depends is called the **declarative multi-agent system**. This article addresses three progressively deeper questions: what exactly is so-called 'agentic AI'? How does one **build** such systems? And how does one combine them with one's own work to determine **what** to build? In the previous article in this series, we proposed the AI4SS (artificial intelligence for social science) technical architecture – 'one core, three repositories, four domains'. The declarative multi-agent system and its pattern language POMASA, the subject of this article, constitute precisely the **'one core' – the intelligent runtime** – the engine that drives the entire AI4SS system. The system's purpose is analytical: it searches, verifies, reasons, and synthesises. That its outputs take the form of text is a consequence of its method, not its

objective. POMASA is not a multi-agent system itself – it is a construction guide for building one: a set of verified architectural patterns that tell you what to build, while the intelligent runtime determines how.

This is not theoretical speculation. The Global South Insights (GSI) project of Tricontinental: Institute for Social Research has already built and operates dozens of declarative multi-agent systems, the largest and most complex of which – the Research Production System – comprises several hundred declarative agents. What follows is drawn from our experience.

1. A New Form of Software – The Declarative Multi-Agent System

When people speak of 'agentic AI', the term is often shrouded in a fog of commercial hype. Once this fog is cleared, what emerges is a straightforward engineering reality: a multi-agent system (MAS) is, at its core, a form of **software** – not magic, not the embryonic stirring of general intelligence, but a new form of software with a clear conceptual framework, architectural principles, and engineering methodology. Grasping this point is the starting point for pulling AI back from hype into material reality.

1.1 From Long Prompts to Multi-Agent Systems: Why MAS Is Necessary

Consider a researcher attempting to use AI for a complex research task. She might write a lengthy prompt, cramming role definitions, research methods, theoretical frameworks, output formats, and quality standards into a single block of text. As the task grows more complex, this prompt swells to 30,000 or even 50,000 words – and then three structural problems become impossible to avoid.

First, **attention dilution**. Even without exceeding the model's context window, large language models confronted with lengthy, detail-laden prompts will simplify their execution autonomously, ignoring details they deem unimportant, rendering output quality unpredictable. Second, **the absence of modular isolation**. Even if the author mentally divides the prompt into 'step one, step two', all content is loaded into a single context, and modifying one section may unexpectedly affect another – much like writing all code in a single enormous function: though divided by comments into sections, it remains a monolith. Third, **maintenance becomes intractable**. Without flexible parameter passing, without dynamic process control, modifying a style guide requires reloading the entire 30,000-word prompt.

These are not hypothetical problems. Any researcher who has tried to encode a complete analytical framework – role definitions, source hierarchies, epistemological commitments, citation standards, style guides – into a single AI conversation has encountered exactly this: the AI begins ignoring citation standards; modifying the analytical framework destabilises the output format; each new conversation requires reloading everything from scratch. The carefully engineered knowledge specification has outgrown the single-conversation container.

For those with software engineering experience, this predicament is familiar. A long prompt is, in essence, text concatenation – role definitions, methodologies, style guides, and other components stitched together into one massive whole and fed to the AI in its entirety. It is akin to reading an entire operations manual aloud to someone from cover to cover, then expecting them to execute every task from memory. Software engineering was in this same stage in its earliest decades – all code written in a single file – and from the structured programming revolution of the late 1960s onwards, spent decades evolving toward today's modular architecture: each module performs a distinct function, can be started and stopped independently, and collaborates through clearly defined interfaces. Current prompt engineering remains stuck at the 'everything in one pot' stage, and what multi-agent systems seek to accomplish is precisely to complete the journey that software engineering has already travelled – giving each module an independent lifecycle and clearly defined boundaries.

The solution: rather than entrusting everything to a single AI conversation, decompose a large task into multiple **agents** – each agent is an independent AI work unit, dedicated to a specific task (such as 'collect literature on a given topic' or 'write analysis according to a specified framework'). Multiple agents, each responsible for its own role and working in coordination, constitute a multi-agent system.

1.2 Core Concepts of MAS

Understanding how multi-agent systems work requires grasping five core concepts.

The **agent blueprint** is a design-time definition document. It is a text file written in natural language – the most common format is Markdown (a lightweight formatting syntax widely used for writing documents), though even a plain text file will do – describing what an agent should do: role definition, input specification, task description, output specification, and quality standards. A blueprint is to an agent what an architectural drawing is to a building – the drawing is not the building itself, but a complete description of it.

The **agent instance** is a runtime execution entity. When a blueprint is 'launched', the system creates an instance with its own independent context window; it understands the task, executes operations, and produces results within its own space. A single blueprint can launch multiple instances simultaneously – just as a single set of blueprints can be used to construct many buildings of identical structure.

The **runtime environment** is the AI platform that executes agents. Unlike traditional programme runtimes, however, an AI runtime (such as Claude Code) is not a passive instruction executor but an **intelligent runtime** – it can understand natural-language intent, autonomously select execution strategies, and make judgements and adjustments when problems arise. A traditional runtime mechanically executes instruction sequences; an intelligent runtime understands intent and intelligently selects execution methods.

The **data bus** is the channel through which agents pass data to one another – in the most common current implementation, this is simply the filesystem. Agent A writes its output to a designated directory; Agent B reads from that directory as input. This seemingly 'low-tech' design embodies a profound consideration: all intermediate products are files that humans can directly read – complete, transparent, and traceable.

Reference data is externalised domain knowledge – methodological documents, theoretical frameworks, disciplinary standards, domain ontologies, and so on. For example, GSI maintains a reference document on the foundations of Marxist theory that is shared across multiple systems as a common analytical baseline (a future article in this series will examine this reference document in detail). Shared by all agents but invariant across tasks, reference data constitutes the knowledge foundation of a MAS and the vessel for institutional knowledge assets.

In practice, each of these five concepts maps onto concrete research activities. Agent blueprints are Markdown files describing tasks such as 'collect and verify fiscal data on a country's extractive sector' or 'analyse distributional impacts using a declared epistemological framework'. Agent instances are the individual AI work sessions launched from these blueprints – one instance collecting data on natural gas contracts, another simultaneously collecting data on mineral royalties. The runtime environment is the intelligent AI platform executing these agents. The data bus is a set of folders – one containing verified data, another containing thematic analyses, a third containing draft sections – each readable and inspectable at any moment. And reference data comprises the institutional knowledge assets assembled by the research team: epistemological framework documents, methodology guides, style standards, and the primary sources that ground every agent's work.

1.3 Declarative vs. Imperative: Why the Declarative Approach Is More Advanced

There are two fundamentally different approaches to constructing multi-agent systems.

The **imperative** approach was the dominant method in pre-2025 frameworks (such as LangChain and AutoGen): agent behaviour is defined through Python code – code that controls *how* the agent acts. This approach requires programmers, imposes a high barrier to entry, demands substantial code, and makes debugging complex.

The **declarative** approach is a new paradigm: agent behaviour is defined through natural-language documents – describing *what properties the result should possess*, while delegating the *how* to the intelligent runtime to determine autonomously at execution time. After Anthropic released Claude Code in 2025, the declarative method rapidly became the prevailing approach – because Claude Code, as an intelligent runtime, natively supports launching and managing sub-agents through natural-language blueprints, rendering imperative frameworks no longer a prerequisite for building multi-agent systems.

This distinction is not a matter of stylistic preference but of architectural choice, and its essence lies in **deferring decisions from design time to execution time**. An imperative system determines at the moment the developer writes the code how to handle a given situation; a declarative system describes only what the result should look like, delegating specific execution strategies to the intelligent runtime to determine autonomously when confronting actual conditions. This is not laziness but an acknowledgement of a material fact – at design time, the developer possesses far less information than the AI system does at runtime. Just as an architect should not mark on the blueprints 'if it rains tomorrow, use Plan B', but should delegate tactical adjustments to the intelligent construction team on site.

The practical difference is stark. Under the imperative approach, a researcher would need a programmer to write Python code specifying, step by step, how to search for data, how to parse the results, how to handle a broken link or a PDF that fails to download. Under the declarative approach, the researcher writes a blueprint in plain English stating: 'Collect fiscal data on the extractive sector from 2015 to 2025, prioritising government budget documents, reconciliation reports, and IMF Article IV consultations; verify each figure against at least two independent sources; flag any discrepancy exceeding 5 per cent.' She describes what the output should look like; the intelligent runtime determines how to achieve it. The barrier between researcher and working system is no longer code – it is the clarity of her own specification.

Quantitative evidence from practice supports this choice. PayPal's large-scale practice in 2025 found that the declarative approach reduced development time by 60%, tripled deployment speed, and decreased agent blueprint size by an order of magnitude. PayPal's engineers concluded: 'Most agentic workflows are composed of common patterns – data serialisation, filtering, retrieval-augmented generation, API orchestration – that can be expressed through a unified declarative language rather than requiring imperative code.'

History provides the best framework for understanding this transformation. From assembly language to high-level languages, from procedural programming to SQL's declarative queries, from manual deployment to 'infrastructure as code' – each elevation in the level of abstraction has dramatically expanded the boundary of *who can participate*. The declarative multi-agent system is the latest milestone on this evolutionary line: it lowers the threshold for 'building AI systems' from 'being able to write code' to 'being able to describe a problem'.

This means: **the builder no longer needs to be a programmer.**

What the builder needs is **domain knowledge** – the capacity to understand problems and to articulate clearly 'what I need'. For researchers and movement leaders in the Global South, this transformation carries a decisive implication: the dependence on software engineers to build one's own AI tools is no longer necessary.

2. Anyone Can Build – Pattern Language as a Construction Guide

The declarative multi-agent system lowers the threshold of *who can build*, but *how to build well* still requires best practices. A builder without experience, confronted with a blank page, still does not know where to begin, how many agents to define, how they should collaborate, or how quality should be assured. Pattern language exists precisely to solve this problem.

2.1 From 'Being Able to Code' to 'Being Able to Describe' – A New Empowerment

The core liberation of the declarative multi-agent system lies in this: what the builder needs to do is not write code but write documents. An agent blueprint is a Markdown file containing role definitions, input specifications, task descriptions, output specifications, and quality standards. Modifying an agent's behaviour means modifying a passage of text – no compilation, no deployment, no debugging.

This means **domain experts can directly participate in defining and modifying the system**. A political economist can read an agent blueprint, understand what it does, and modify its analytical framework – in the natural language she knows well, rather than in Python, which she does not. Research methodology can be translated directly into system behaviour.

This carries particular significance for the Global South. Under the traditional AI development model, researchers must 'translate' their requirements for software engineers, who then implement them in code. In this translation process, intent is inevitably distorted and simplified. The declarative system eliminates this translation step – the researcher herself is the builder of the system.

2.2 Pattern Language: Executable Best Practices

But freedom does not automatically produce excellence. New builders still need to know 'what a good system looks like'.

Pattern language is the classic answer to this question. It originates in architecture – Christopher Alexander proposed in his seminal 1970s work *A Pattern Language* a set of practice-verified design patterns, each solving a recurring design problem, with patterns forming an organic network of relationships that together guide builders in creating high-quality works.

Consider one of Alexander's original examples: Pattern #159, 'Light on Two Sides of Every Room'. Alexander observed that when people can choose, they always gravitate toward rooms with windows on two sides, while rooms with only one window are neglected – because single-sided light creates sharp contrasts and glare that instinctively cause discomfort. The pattern's guidance is therefore: **when arranging each room, ensure it has exterior walls on at least two sides where windows can be placed, allowing natural light to enter from more than one direction.** This is a 'pattern' – it identifies a recurring problem (single-sided lighting causes discomfort), provides a verified solution principle (light on two sides), and does not prescribe specific construction methods. Software engineering successfully imported this thinking in the 1990s – the Gang of Four design patterns remain required study for every programmer.

POMASA is a pattern language that inherits this tradition from Alexander. Consider one of POMASA's examples: Pattern STR-02, 'Filesystem Data Bus'. When multiple agents collaborate, a central question is: how do they pass information to one another? One could use a database, a message queue, or various sophisticated technical solutions – but intermediate results then become invisible to humans, locked within the system's internals. This pattern provides the principle: **use the simplest possible mechanism – files and folders.** Agent A writes its output as a file in a designated directory; Agent B reads from that directory as input. All intermediate products are documents that a human can open and read directly. If anything goes wrong at any stage, opening the corresponding folder reveals exactly what each agent produced – no black box.

In the AI era, pattern language undergoes a critical transformation. Traditional pattern languages were written for humans to read – humans understand the pattern and then translate it into code themselves. But in an intelligent runtime environment, the pattern language itself is written in natural language, and the AI runtime can understand natural language – therefore, **the pattern language itself becomes executable**. One can hand a set of patterns to an AI system, and it can directly generate a working multi-agent system from them.

This redefines what 'open source' means. Traditional open source meant publishing code for others to run.

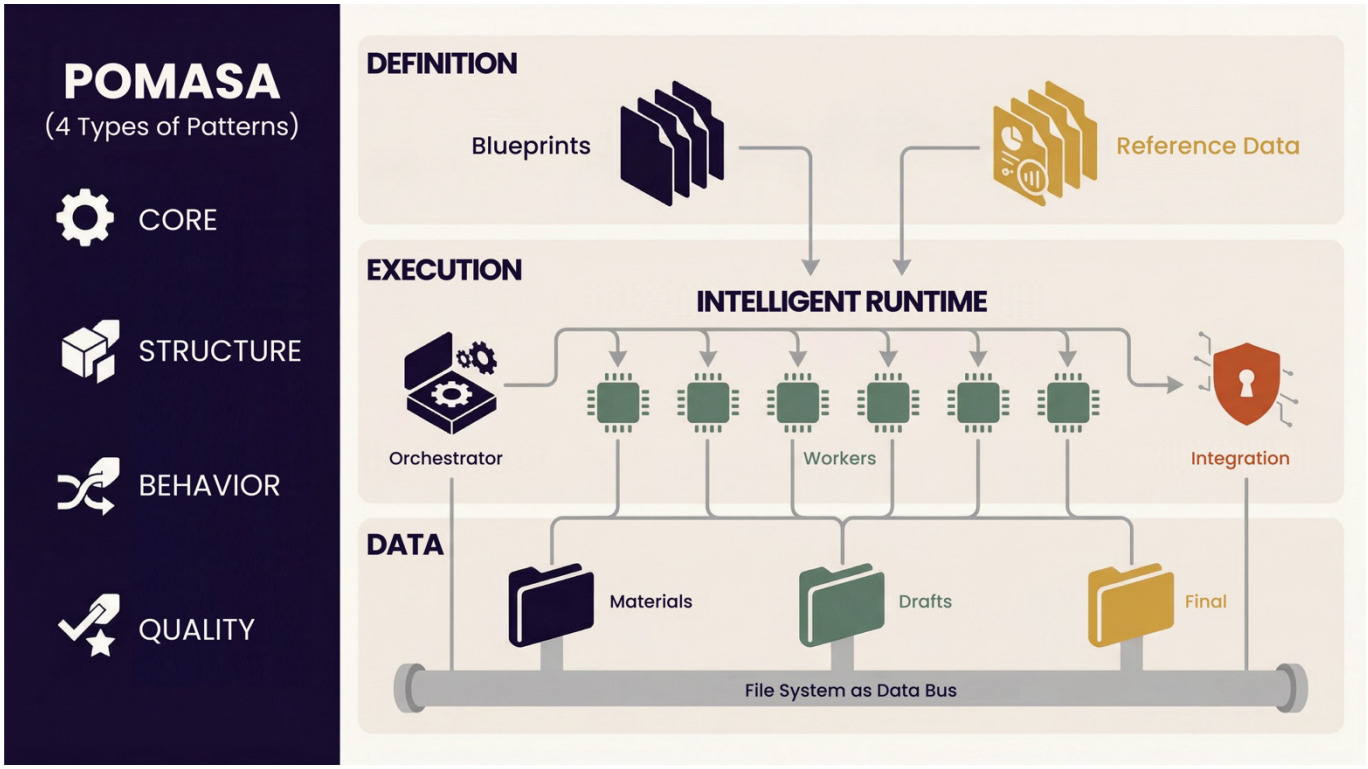
In the AI era, publishing a pattern language is equivalent to publishing source code – because others can hand the patterns to an AI system and reconstruct an equivalent architecture.

2.3 POMASA: Our Pattern Language

POMASA (Pattern-Oriented Multi-Agent System Architecture) is precisely such a pattern language. It is not a specific software system but a set of architectural patterns refined from practice and verified through deployment.

POMASA comprises twenty patterns in four categories:

- **Core patterns** (COR, 2 patterns): Prompt-Defined Agent (COR-01) and Intelligent Runtime (COR-02) – the foundation of everything.
- **Structure patterns** (STR, 9 patterns): Reference Data Configuration, Filesystem Data Bus, Workspace Isolation, Business-Driven Agent Design, Composable Document Assembly, Methodological Guidance, and others – defining the system's skeleton.
- **Behaviour patterns** (BHV, 6 patterns): Orchestrated Agent Pipeline, Faithful Blueprint Instantiation, Parallel Instance Execution, Progressive Data Refinement, and others – defining how the system operates.
- **Quality patterns** (QUA, 3 patterns): Embedded Quality Standards, Embedded Quality Standards, Layered Quality Assurance, Verifiable Data Lineage – defining how the system ensures reliability.



POMASA architecture overview: from blueprint definition to intelligent runtime execution, connected through the filesystem data bus

Each pattern is classified by necessity into three levels: **Must**, **Recommended**, and **Optional**. The minimum viable configuration requires only six must-have patterns to get started; ten recommended patterns cover the majority of scenarios; additional optional patterns can be introduced progressively according to actual needs.

A new builder need not absorb all twenty patterns before beginning. The six must-have patterns are sufficient to transform a single overloaded conversation into a modular system: defining agents in Markdown (COR-01), running them on an intelligent runtime (COR-02), passing data between them through folders (STR-02), isolating each agent's workspace (STR-03), embedding quality standards directly into each blueprint (QUA-01), and orchestrating agents in a pipeline (BHV-01). As confidence grows, recommended patterns can be introduced progressively – parallel execution to research multiple dimensions simultaneously, layered quality assurance to separate the agent that writes from the agent that reviews.

This pattern language was not designed in the abstract. It was refined from real production systems – starting from five patterns and gradually growing to twenty through repeated system construction and fault resolution. Every pattern corresponds to a mistake once made or a design challenge repeatedly encountered.

2.4 Start Now: Three Steps to Begin

POMASA is not an armchair exercise. If you wish to try it now, only three steps are required.

The prerequisite is an intelligent runtime that supports sub-agent invocation – such as Claude Code, Cursor, or Cline. After installing the runtime of your choice, run npx skills add eXtremeProgramming-cn/pomasa in the command line to install the POMASA skill (this command supports all the runtimes mentioned above). Then, describe in natural language what you wish to research – for example: 'Help me create a multi-agent research system to analyse digital economy policies in Southeast Asia.' The AI will guide you in refining your requirements, recommend an appropriate combination of patterns based on your description, and then automatically generate the complete set of system files.

No single line of code is required throughout the entire process. The generated system is a set of Markdown files – agent blueprints that you can read, understand, and modify. POMASA itself is open-sourced under the Apache-2.0 licence. Moreover, this is not a demonstration piece – it is the same pattern language that GSI uses in its daily work.

2.5 Evidence from GSI: The Research Production System in Practice

GSI's **Research Production System** (RPS) is the most thorough validation of POMASA in practice. RPS comprises several hundred declarative agents and hundreds of quality gates, the overwhelming majority blocking – 100% declarative, without a single line of Python orchestration code. Every agent is a Markdown file.

One of RPS's flagship products is the Country Conjuncture Report – a systematic study of the political-economic situation in different countries. Taking the report on an African country as an example, this report passed through a multi-stage pipeline – from literature acquisition and ingestion, conjuncture analysis, research planning, deep research (with multiple topics per batch and several agents working in parallel), comprehensive writing, quality review, publication preparation, to summary generation. The entire process handled over a hundred input documents across dozens of research topics and produced a comprehensive report containing thousands of citations. The system was completed in a matter of days. Completing a country conjuncture report of equivalent quality through traditional methods would require six to twelve months. The efficiency gain amounts to 100 to 200 times.

But efficiency is not the only measure. Quality assurance is the central question that any declarative MAS must answer – because AI systems 'hallucinate' (research has shown that over half of the academic citations generated by ChatGPT are fabricated, and even GPT-4 still exhibits an 18% fabrication rate). RPS addresses this through hundreds of quality gates and dozens of producer-validator pairs: the agent that generates content cannot review its own output; validation must be

carried out independently by a separate agent starting from zero. The system also incorporates a multi-layer fact-checking process with character-level precision – in a completed research report, hundreds of endnotes achieved zero fabricated citations. In external evaluation, the system scored highly, with the quality assurance dimension receiving a perfect score and the system assessed as 'a production-ready system with significant innovation'.

This is not theory. This is a production system already in operation, generating real research outputs.

3. A Complete Example – From One Sentence to a Research Report

The preceding two sections set out the principles of declarative MAS and the pattern system of POMASA, but the reader may well ask: what does all of this look like in practice? Let us walk through the complete process, from a single paragraph of requirements to a finished research report containing 85 citations.

A reader who had been following the US-China technology competition came across seven articles on the subject online, covering chip wars, rare earth contestation, renewable energy, and tariff policy. Each article addressed a different facet of the issue, but he wanted a comprehensive overview that brought these scattered analyses together, helping him understand the complex landscape systematically. His core requirement, entered into the POMASA user input template, occupied a single paragraph: 'The US-China chip tech war – covering chips, AI, robotics, energy, rare earths, and related supply chain issues. Bring this all together and complement it with full supply chain analysis, including China's manufacturing prowess and supply chain ascendancy in adjacent areas such as electric vehicles, batteries, solar panels, and precision tools.' He provided the seven articles to the system as reference materials.

He fed this requirement into the POMASA generator. The generator guided him through several key parameters: data source preferences (priority to policy documents, peer-reviewed research, and credible industry reports, with attention to non-English sources particularly Chinese); output format (research overview, 5,000 to 10,000 words); and quality assurance level (the strictest tier available). The generator then automatically selected thirteen of the twenty available patterns and produced a complete multi-agent research system – seven agent blueprints, a full set of reference data, methodology documents, and a directory structure, all consisting of readable, modifiable text files.

The generated system operates as an eight-stage pipeline. The narrative architect reads all seven reference articles and designs the narrative structure along with a detailed research plan. Deep researchers are assigned by topic and execute in parallel – each topic launches an independent agent instance that collects materials from the internet. The data verifier checks every item of collected data individually: not a sample but 100 per cent verification, a mandatory requirement of POMASA's quality patterns. Analysts distil arguments from the verified materials, organised by theme. Section writers draft each section according to the designated style guide. Finally, the quality auditor, operating as an independent role, conducts a comprehensive review from scratch.

From filling in the user input template to generating the complete multi-agent system took roughly half an hour. The system then ran – deep research, data verification, analysis, writing, and quality audit – for several hours, investigating hundreds of verified sources from the internet in the process. The final output was a report entitled 'Breaking the Siege: China's Technological Ascent and the Unravelling of US Tech Hegemony'. Exceeding 11,000 words and containing eighty-five inline citations with URLs, it covered semiconductor sanctions and indigenous breakthroughs, AI chip competition, rare earth countermeasures, renewable energy dominance, 'dark factory' automated manufacturing, and implications for the Global South. The quality audit scored highly, with argumentation and Global South perspective both receiving perfect marks. The auditor simultaneously produced a concrete revision checklist – identifying areas for refinement and citations requiring further cross-verification – while assessing the core analysis as 'publication-ready'.

The pattern, not the particular instance, is what matters. The same process applies whether the domain is technology competition, extractive industries in East Africa, land reform in South Asia, or labour conditions in Central America. The requirements change; the domain knowledge changes; the system architecture remains the same: requirements in, system out, research produced – with every step traceable.

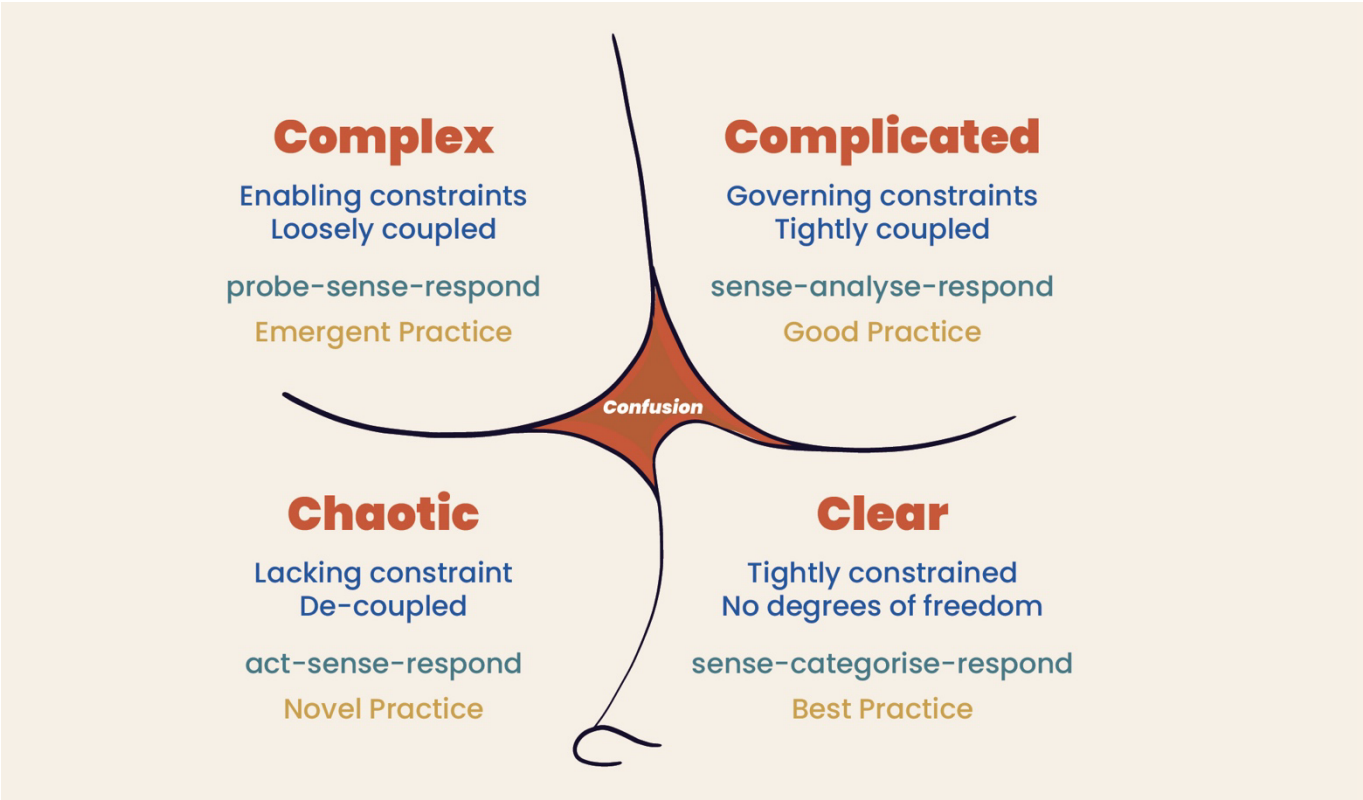
This was not a flawless output – the very purpose of a quality audit is to identify deficiencies. Final judgements, refinements, and structural revisions still require human involvement; this is the practical meaning of 'human in the loop'. What warrants attention, however, is the process itself: one reader's single paragraph of requirements and seven reference articles, combined with POMASA's pattern system and without any programming, generated a complete research system that autonomously produced a substantial report with extensive traceable citations. This is precisely the leap from consumer to builder described at the outset of this article – not a metaphor, but a fact we have witnessed firsthand.

4. What to Build – Design Guided by the Nature of the Task

'How to build' is a technical question; 'what to build' is a strategic one. A common mistake is this: once one has the declarative multi-agent system as a hammer, every task begins to look like a nail. Tasks of fundamentally different natures require fundamentally different system designs. The blind pursuit of full automation is the most common and most dangerous trap.

4.1 The Cynefin Framework: Understanding the Nature of Tasks

Dave Snowden's Cynefin framework provides a decision tool for judging 'what kind of MAS suits this task'. It classifies tasks by the clarity of their causal relationships into four domains:



Cynefin framework

The Clear domain – causal relationships are transparent and the path is known. Examples include batch file downloads, format conversions, and data cleaning. These tasks are suited to fully automated pipelines requiring no human intervention.

The Complicated domain – causal relationships exist but require expert judgement, and analytical frameworks can be determined in advance. An example is conducting country assessments according to an established indicator system. These tasks are suited to parallelisable analytical pipelines: multiple agents simultaneously collect and analyse data along different dimensions, then consolidate results.

The Complex domain – causal relationships can only be identified in retrospect, and critical insights must come from humans. An example is conducting a political-economic analysis of the class structure of a given country – without understanding the historical context of racial dynamics, data analysis alone will miss the mark entirely. These tasks demand **iterative human-machine collaboration**: AI conducts exploratory research and material preparation, the human provides insights and theoretical frameworks, and AI then performs deep analysis under the human's framework guidance.

The Chaotic domain – the required information does not exist in any accessible data source and must be created through human action. An example is fieldwork conducted in factories and villages to gather primary materials – information that does not exist on the internet, has no archival record, and is beyond the reach of AI. In these tasks, AI can serve only as an auxiliary tool for mechanical work such as transcription, note organisation, and background material retrieval.

Most real research projects span all four domains simultaneously, and recognising this is essential to designing an effective system. Consider a study of why extractive wealth has not produced broad-based development in a given country. Translating government budget documents and cleaning fiscal datasets are Clear-domain tasks – fully automatable. Collecting and analysing extractive revenue data along established dimensions (royalty rates, corporate tax receipts, production-sharing terms) is a Complicated-domain task – agents can work in parallel across commodities using a predetermined analytical framework. But the central analytical question – identifying the specific configuration of class forces, state-capital relations, and international financial pressures that produce this outcome – belongs to the Complex domain: it requires the researcher's own political-economic judgement, something AI cannot supply. And the experiences of communities displaced by mining operations – the anger, the broken promises, the local political dynamics – belong to the Chaotic domain, accessible only through fieldwork that no AI system can perform. A well-designed system automates the first two domains, structures iterative human-machine collaboration for the third, and limits AI's role in the fourth to transcription and background retrieval.

A simple 'three-question decision tree' can help make the determination: Does the information needed to complete this task already exist? (No – Chaotic domain.) Is the analytical path from information to conclusions clear? (No – Complex domain.) Does it require expert-level judgement? (Yes – Complicated domain; No – Clear domain.)

4.2 Domain Degradation: The Evolution of the Research Process

The deeper insight that the Cynefin framework reveals is this: **the research process itself is a progression from the Chaotic/Complex toward the Complicated/Clear.**

A research project typically begins in the Chaotic or Complex – one is uncertain what to study, which framework to employ, or where to begin. As understanding deepens, the injection of human frameworks degrades Complex tasks into Complicated ones: once analytical dimensions and assessment criteria are established, what remains is structured data collection and analysis, which AI can accomplish efficiently. As methods mature, Complicated tasks further degrade into Clear ones: assessment processes become codified and can be repeated in fully automated fashion.

A well-designed multi-agent system should support this evolution – progressively increasing the degree of automation as understanding deepens. This is also the most important design warning: **do not force-automate every task.** Forcing automation on Complex or Chaotic domain tasks typically produces large volumes of shallow, insight-free content. Recognising AI's boundaries is as important as recognising its capabilities.

4.3 GSI's Practice: Different Configurations for Different Tasks

GSI's actual systems embody distinct configurations for each of these four domains.

Clear domain: the fully automated pipeline. A researcher has a batch of documents in a language she cannot read and needs them translated into one she can. This task once required professional translators; now it can be handed entirely to AI – the inputs are defined, the processing rules are standardised, the output format is clear. The characteristics of such tasks are: information locations are known, retrieval paths are clear, processing methods are standardised – they can be entrusted entirely to AI for automated completion.

Complicated domain: the parallelisable analytical pipeline. The Digital Sovereignty Index (DSI) assessment is the representative case in this domain. The assessment framework encompasses four dimensions and sixteen indicators, elaborated into thousands of assessment criteria. For each country, the system collects hundreds of pieces of raw evidence, which are filtered through deduplication and quality checks to yield high-quality items. The complete pipeline produces a structured assessment report in a matter of hours – assessments have been completed for all eleven BRICS countries. The characteristics of such tasks are: the analytical framework is already established (designed in advance by human experts), and what remains is large-scale structured data collection and analysis, which AI can accomplish efficiently in parallel.

Complex domain: iterative human-machine collaboration. RPS's country conjuncture studies represent this domain. The system employs a universal nineteen-dimension analytical framework (spanning material foundations, class forces, and contradiction analysis), but when applying this framework to a specific country, the critical step is 'contextualisation' – identifying the principal contradictions specific to that country, something AI cannot accomplish automatically. Take South Africa as an example: without understanding the historical legacy of settler colonialism, without understanding how racial dynamics have shaped the particular form of class contradictions in the country, AI can collect as much data as it likes and still miss the essential point entirely. Such tasks demand human-machine collaboration – AI processes thousands of source materials and produces dozens of thematic research reports; the human, on this foundation, provides critical analytical judgements (such as determining that South Africa requires the addition of 'the legacy of settler colonialism' as a country-specific analytical dimension); AI then, under the human's framework guidance, weaves the fragmented research into a coherent analytical narrative. The injection of the human framework is precisely the step that degrades a Complex task into a Complicated one.

Chaotic domain: AI as an auxiliary tool. Fieldwork conducted in factories and villages to gather primary materials – information that does not exist on the internet, has no archival record, and is beyond the reach of AI. In this domain, the human leads the entire process, and AI undertakes only mechanical auxiliary work such as audio transcription, note organisation, and background material retrieval. Yet even here, AI's assistance has value – one doctoral researcher, after using AI to process interview recordings, 'discovered in the transcripts information that he had not heard or noticed during the interviews themselves'.

5. Implications for the Global South – The Convergence of Declarative MAS and AI4SS

The preceding sections addressed the hammer itself – why we chose this hammer, how to forge it, what it produces in practice, and what kind of nails it drives. This section must answer a different question: **why does this hammer matter especially for the Global South?**

The answer lies not in the technology itself but in the structural alignment between the technology and the concrete conditions facing the Global South.

5.1 The Structural Obstacles Confronting the Global South

Before discussing AI's potential, we must first confront the structural obstacles facing the Global South.

The class asymmetry of knowledge production. Scholars who serve the interests of capital possess abundant resources, time, and energy to produce knowledge – yet this knowledge serves not the interests of the people but functions as an instrument of exploitation and oppression. At the same time, cadres of popular movements lack both resources and time. They cannot dedicate extensive hours to reading, research, and writing. The production and consumption of knowledge is monopolised along class lines – a cruel irony.

The contradiction between information volume and cognitive bandwidth. Lenin demanded in *The Tasks of the Youth Leagues* (1920) that Marxists 'enrich their minds with a knowledge of all the treasures which humankind has created', and the Marxist method of totality requires an integrated understanding spanning economics, politics, culture, and ideology. But these treasures of knowledge are too vast in quantity, too rapid in change, and too specialised in their subdivisions, far exceeding the limits of individual cognition. The method of totality thus becomes an ideal that cannot be put into practice.

Language barriers and resource barriers. The greatest volume of knowledge is produced in English, and translation resources are acutely scarce. Academic resources and databases charge exorbitant fees, and progressive media lack both resources and personnel. For a researcher in the Global South to access the data and literature that a scholar in the Global North takes for granted often requires a disproportionate expenditure of time and money.

The North-South infrastructure asymmetry. AI systems, publishing institutions, and citation indexes are concentrated in the North. Researchers in the Global South are evaluated by Northern standards, publish in Northern venues, and have their work framed by Northern theoretical concerns. The asymmetry of digital infrastructure – from computing power to network connectivity to electrical supply – means that the AI wave risks widening rather than narrowing the North-South gap.

5.2 A Threefold Liberation

Drawing together the foregoing analysis, the declarative multi-agent system offers the Global South a threefold structural liberation.

From technology consumption to technology construction – epistemological sovereignty. Declarative MAS combined with a pattern language such as POMASA enables the Global South to build its own tools using its own domain knowledge, without waiting for the benevolence of Silicon Valley. In a declarative system, the blueprint is the source code and methodology is the programme – and what the Global South possesses is precisely its own methodology and domain knowledge. 'Publishing a pattern language is equivalent to publishing source code' signifies a new form of open source – we are not merely using the North's open-source tools; we are creating the South's own open-source assets.

From knowledge monopoly to the democratisation of knowledge – capability multiplication. A small team of a few people, combined with a declarative MAS, can produce outputs that previously required a large research institution – several hundred agents, thousands of citations in a single country report, efficiency gains of 100 to 200 times. This capability multiplication is of particular consequence for the Global South: it enables resource-constrained progressive institutions, trade union research departments, and small independent media to conduct rigorous, evidence-based research and content production. The capacity for knowledge production, once monopolised, is being returned.

From mechanical toil to intellectual creation – the liberation of the human being. AI takes over the mechanical labour of desk work – searching, organising, transcribing, preliminary analysis – liberating humans from the heavy burden of information processing. But the purpose of this liberation is not leisure; it is to allow humans to return to the field, to the grassroots, to the masses, to do what only humans can do: generate insights, point the way forward, and render final judgements. What Marx called the unity of theory and practice – *praxis* – should be completed as a closed loop within the same person. In our practice, we have come to recognise with ever greater clarity: 'Human thought, human experience, human insight – these are the most essential, most luminous elements in the entire process of knowledge and content production.' AI is not intended to replace these – quite the contrary, the entire value of AI lies in allowing the human mind to focus on those most essential tasks that AI cannot perform.

The arc described across this article – from the structural limitations of a single AI conversation, through the modular architecture of multi-agent systems, to the pattern language that guides their construction – traces a journey from consumption to construction. A researcher who begins by pasting her methodology into a chat window and ends by building a system of hundreds of coordinated agents has not merely adopted a new tool. She has reclaimed the capacity to shape her own means of knowledge production. Her epistemological framework becomes reference data; her methodology becomes an agent blueprint; her quality standards become embedded quality gates. She does not code; she describes. And what she describes is not a wish but a specification – grounded in her training, her domain knowledge, and her political commitment to the communities whose conditions she studies. The system she builds belongs to her institution; the methodology it encodes is her own; the knowledge it accumulates compounds from project to project.

5.3 Restitution: From 'Users' Back to Makers

The significance of declarative MAS extends beyond efficiency gains or capability multiplication – it touches upon a deeper historical process.

For the past four decades, the information technology industry has systematically transformed the vast majority of people from makers of tools into mere 'users'. As Bonnie Nardi documented in *A Small Matter of Programming*, the graphical user interface severed the continuity between using and creating; software commodification closed off the pathways to building; developer communities evolved into knowledge guilds; ever-escalating complexity raised the barrier higher and higher. In the end, it was not merely skills that disappeared – the very concept of 'I can build tools for myself' vanished from people's language and imagination. When something cannot be spoken, it cannot be thought. The dispossession of capability thus became permanent and self-sustaining.

The declarative multi-agent system reverses this undercurrent. In this architecture, blueprints are Markdown rather than code; domain knowledge is the programme; methodology is the software. Building an agent system capable of autonomous deep research no longer requires permission from the programmer's guild – what it requires is the knowledge in the domain expert's mind: along which dimensions to analyse a problem, from which sources to gather information, which epistemological framework to guide judgement. And this knowledge is precisely what the progressive forces of the Global South have never lacked.

This is not endowment; it is restitution.

The capacity for ordinary people to build their own information tools existed from the beginning – researchers in the Unix era wrote their own scripts; users in the personal computer era booted their machines to a BASIC command line. This capacity is not a novelty brought about by technological progress; it is an old right that was systematically taken away by four decades of industrial evolution. The declarative multi-agent system gives it back.