AI and cREXX

Lessons from building with agents, then calling and supervising LLMs from Rexx.

- Agents need boundaries, tests, and feedback
- LLMs are callable from cREXX
- Rexx can supervise AI pipelines

Paper record

A paper-record version of the live AI/cREXX symposium session.

The Three Part Story

The talk moved from project practice to runtime capability to orchestration.

1. Building with AI

Focused docs, small task ownership, reproducible tests, debugging tools.

2. Calling AI from cREXX

HTTP, TLS, JSON, provider wrappers, and ADDRESS environments.

3. Orchestrating AI

CognitivePipelines with cREXX scripts as deterministic control nodes.

Session promise

The claim was not that AI writes correct code. It was that AI can move fast when the project makes correctness observable.

What Changed In Practice

The useful pattern emerged only after the work became legible to agents.

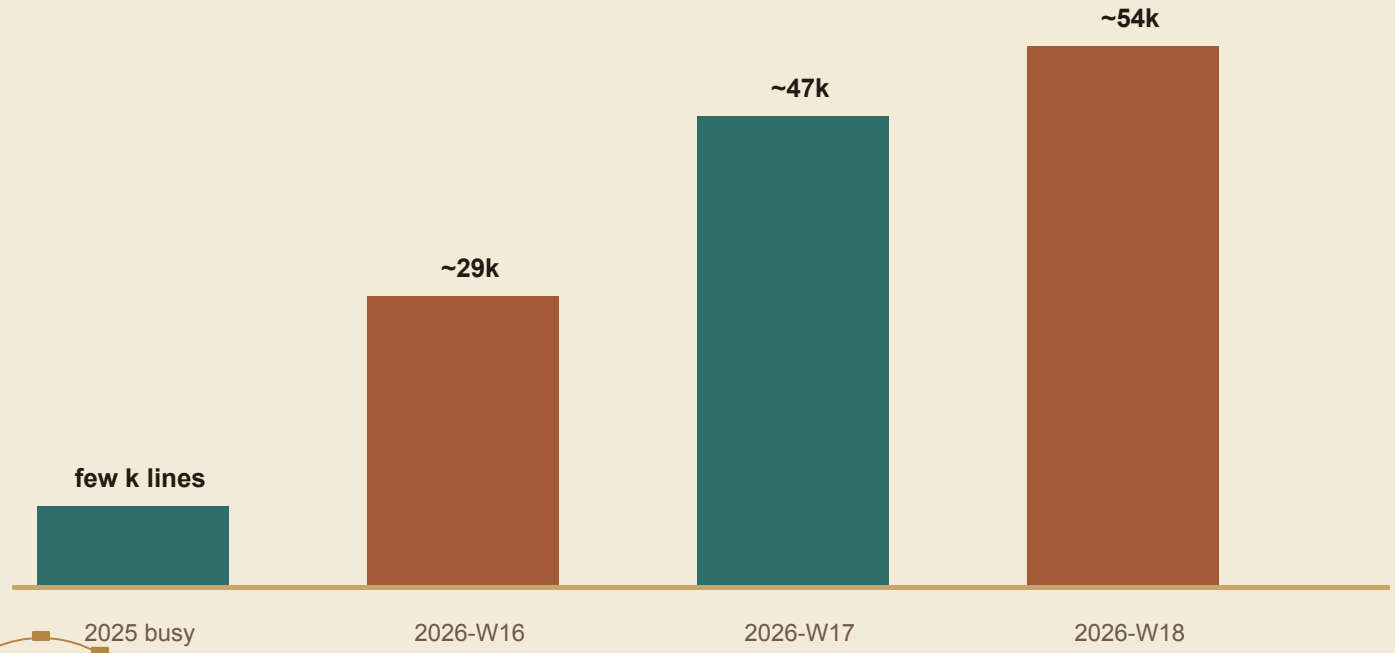
- Broad requests produced broad patches
- Focused docs named the source of truth
- Changes now start from a reproducer or test
- Command-line debugging lets the agent loop
- Human review steers design, naming, and scope

The shift

From AI as a typist to AI as a fast implementer working inside a strict lab bench.

A Rough Productivity Signal

Not a formal study, but the git history showed focused bursts getting much larger.



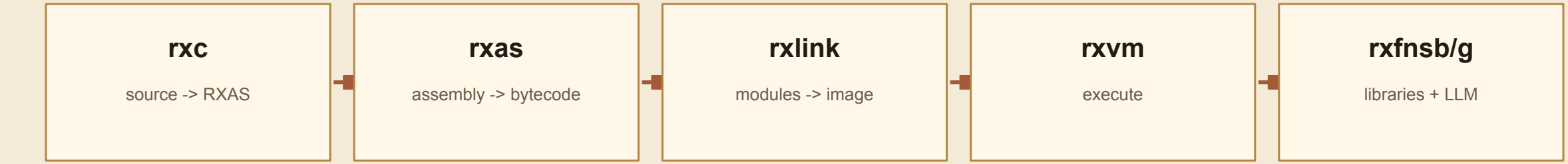
Interpretation

Line counts include docs, goldens, and generated changes. The useful claim is practical throughput: roughly 2x to 4x in focused bursts when tests and architecture were already in place.

Examples in those bursts included ADDRESS environments, crexxsaa, HTTP/TLS/LLM providers, inlining hardening, release packaging, and THE/DSLSH integration.

Agents Worked Because cREXX Has Strong Boundaries

The project gave the model a map and a definition of done.



Additive path

Add a provider, fixture, ADDRESS environment, or doc rather than rewriting a subsystem.

Guardrails

Language-design changes need approval; compiler bugs start with a reproducer.

Feedback loop

Verbose debugging goes to temp logs; tests decide when the loop is done.

The Guardrails That Mattered

Tests made feedback cheap; debugging tools made iteration possible.

Surface	What it caught or explained
parser / grammar fixtures	syntax drift and edge cases
golden RXAS outputs	compiler emission regressions
runtime functional tests	VM and library behaviour
native callback host tests	crexxsaa integration
provider parsing tests	HTTP/JSON/LLM response shape changes
rxc -d2 / -d3, rxas -n, rxdas	AST, symbols, assembly, bytecode inspection

This Matches The Wider Pattern

The local lessons line up with emerging agent practice.

- Simple composable workflows before autonomy
- Clear tools and transparent command output
- Stable test environments and thorough regression checks
- Human design approval for language and architecture
- Agent-created docs to preserve context

References used in the talk

Anthropic agent guidance; Anthropic eval guidance; SWE-agent research on the agent-computer interface; DORA 2024 on AI and delivery discipline.

LLM Support In cREXX Is Built From Ordinary Pieces

The provider code is Rexx code, not hidden host magic.

rxsocket
socket handles

TLS
OS trust stores / OpenSSL / SChannel

rxhttp
HTTP/1.1 framing

rxjson
request + response JSON

rxfnsg.llm
provider layer

Providers

Ollama, OpenAI Responses, Anthropic Messages, Gemini generateContent.

Calls

generate(), generateJson(), extractText(), status(), error().

Diagnostics

Non-2xx bodies and parse failures remain visible to Rexx code.

Direct LLM Client Shape

A clean object call builds the request body and extracts provider text.

12-direct-llm-client.rexx

```
prompt = "Reply with one short sentence."
client = .llm("demo-model", "127.0.0.1", 11434, 1000)

body = client.buildBody(prompt)

fake_json = '{"model":"demo-model","response":"cREXX can supervise an
LLM.","done":true}'
answer = client.extractText(fake_json)

say answer
say "status:" client.status()
```

output

```
Direct cREXX object call
client = .llm(model, host, port, timeout)

Request body built in Rexx
{ "model": "demo-model", ... }

Text extracted from provider JSON
cREXX can supervise an LLM.
status: 0
```

ADDRESS LLM Makes The Call Rexx-Native

The provider becomes a command environment with host-variable input and output.

13-address-llm-envelope.rexx

```
prompt = "Say hello to Rexx programmers."
body = ""
status_text = ""

address llm_gpt_4_1
"BODY :prompt INTO ${body}"
"STATUS INTO ${status_text}"

say "driver:" addresscall("llm_gpt_4_1","driver")
say "model: " addresscall("llm_gpt_4_1","model")
say body
```

output

```
ADDRESS environment: LLM_GPT_4_1
driver: openai
model:  gpt-4.1
status: 0

The prompt is a host variable,
not accidental string interpolation.

Generated request body, no network needed.
```

Provider Routing Is Just A Naming Contract

One ADDRESS class can claim a family of model-shaped environments.

Environment	Driver	Model intent
llm_gpt_4_1	OpenAI	GPT-4.1
llm_gpt_4o	OpenAI	GPT-4o
claude_sonnet_4_5	Anthropic	Sonnet
gemini_2_5_flash	Gemini	Flash
gemma4_latest	Ollama	local Gemma
llm_mistral_latest	Ollama	local Mistral

Demo posture

Network calls can be live, but the deck kept safe offline previews for reliability.

Validation And Retry Is Where Rexx Shines

The model can be creative; the Rexx script can be stubborn.

16-validate-loop.rexx

```
do attempt = 1 to 3
  answer = candidate(prompt, feedback, attempt)
  say "Attempt" attempt "->" answer

  if strip(answer) = "42" then do
    accepted = 1
    leave
  end

  feedback = "Return only the digits."
end
```

output

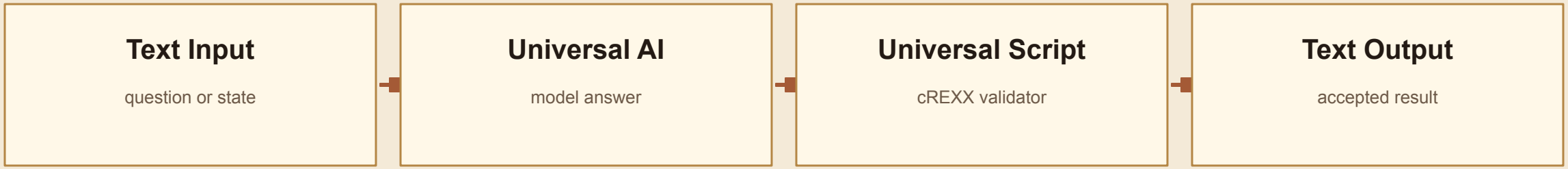
```
CREXX supervises the non-deterministic worker.
prompt: Return only the integer result of 6 * 7.

Attempt 1 -> The answer is forty-two.
Rejected
feedback: Return only the digits.

Attempt 2 -> 42
Accepted answer
42
```

CognitivePipelines Uses cREXX As A Control Node

This part moved to screen-share, but the shape is simple enough for the paper record.



ADDRESS PIPELINE

Scripts read input, write output, log, and return errors through the host environment.

Deterministic gate

The cREXX node validates length, topic, refusal patterns, and required concepts.

Bigger graph

A Transform Scope can retry when the validator rejects the model answer.

The Pipeline Script Is Plain Rexx

The graph is visual; the supervision remains readable text.

21-address-pipeline-script.rexx

```
address pipeline "INPUT" expose cp_input[]

do i = 1 to cp_input.0
  cp_output[i] = "CREXX saw: " || cp_input[i]
end

message = "Processed " || cp_input.0 || " item(s)"
address pipeline "LOG :message" expose message

address pipeline "RETURN" expose cp_output[] cp_log[] cp_errors[]
```

output

```
Pipeline log:
Processed 1 item(s)

Output:
CREXX saw: Rexx is useful as a control language for LLM tools.

Errors:
(none)
```

The AI Takeaway

AI made engineering discipline more valuable, not less.

- Agents accelerated work when boundaries were clear
- Regression tests turned model mistakes into normal debugging
- cREXX can call LLMs through libraries and ADDRESS environments
- CognitivePipelines shows Rexx supervising AI inside a graph
- The next design question is how simple the ADDRESS LLM surface can become