Cognitive BASIC: An In-Model Interpreted Reasoning Language for LLMs

Oliver Kramer

Computational Intelligence Group
Department of Computer Science
University of Oldenburg, Germany
oliver.kramer@uni-oldenburg.de

Abstract. Cognitive BASIC is a minimal, BASIC-style prompting language and in-model interpreter that structures large language model (LLM) reasoning into explicit, stepwise execution traces. Inspired by the simplicity of retro BASIC, we repurpose numbered lines and simple commands as an interpretable cognitive control layer. Modern LLMs can reliably simulate such short programs, enabling transparent multi-step reasoning inside the model. A natural-language interpreter file specifies command semantics, memory updates, and logging behavior. Our mental-model interpreter extracts declarative and procedural knowledge, detects contradictions, and produces resolutions when necessary. A comparison across three LLMs on a benchmark of knowledge extraction, conflict detection, and reasoning tasks shows that all models can execute Cognitive BASIC programs, with overall strong but not uniform performance.

1 Introduction

Recent work on *cognitive prompting* [1] has shown that LLMs can be guided toward more reliable reasoning when prompts explicitly reflect cognitive processes such as goal decomposition, declarative and procedural knowledge extraction, or conflict handling. These approaches move beyond unstructured text generation by imposing cognitive orientation on the reasoning steps themselves. However, they still rely on implicit execution: the model decides how to follow the instructions, and intermediate cognitive states remain informal and difficult to audit.

Cognitive BASIC takes the next step in this direction by enforcing structured reasoning through a minimal in-model programming language. Instead of describing reasoning procedures at the prompt level, Cognitive BASIC executes them through a BASIC-style, line-numbered program interpreted entirely by the LLM. An interpreter file, written in natural language, defines the semantics of each command, the memory manipulation rules, and the logging behavior. Programs operate on a compact working memory containing declarative knowledge (what is known), procedural knowledge (how to act or reason), detected contradictions, and reconciled resolutions. Each instruction updates this memory state explicitly, producing a transparent, auditable reasoning trace.

This design connects two traditions: the transparency aims of cognitive prompting, and the explicit control flow of early programming languages such as BASIC [2]. While prior prompting frameworks, such as Chain-of-Thought [3], ReAct [4], or modular cognitive prompts [1], encourage structured steps, they lack an executable semantics. Classical cognitive architectures including ACT-R [5] and SOAR [6] separate declarative and procedural memory under symbolic control, and recent agentic systems such as OpenCog Hyperon [7] or MemGPT [8] offer persistent memory for extended reasoning. Yet these approaches rely on external engines or customized environments.

2 Cognitive BASIC Language and Interpreter

Cognitive BASIC adopts the simplicity of early BASIC to structure reasoning inside a language model. Programs consist of short, numbered lines executed sequentially unless redirected by control flow. The interpreter, defined entirely in natural language, runs within the model and updates a compact memory state after each instruction.

2.1 Execution Model

The interpreter follows deterministic BASIC-style semantics [2]. Lines execute in ascending order, with conditional branching through `IF ... THEN <line>` or direct jumps using `GOTO <line>`. After each command, the interpreter applies the operation to the current memory, prints a concise log entry, and proceeds to the next instruction. Execution terminates on `END`, producing a final structured memory state that summarizes all reasoning steps.

2.2 Memory Schema

Cognitive BASIC maintains a compact memory structure that serves as the model's internal mental model during program execution. The variable `working` stores the current scenario text or intermediate content and acts as a short-term buffer for each instruction. The fields `declarative` and `procedural` represent the two central forms of cognitive knowledge: factual propositions describing what is true, and operational rules describing how to act or reason. Together, they provide the basic components of a structured mental model.

Contradictions discovered during execution are recorded in `conflicts` as simple string pairs of the form "A || B", making cognitive inconsistencies explicit rather than implicit. When a conflict is repaired, the resulting reconciled statement is stored in `resolution`, documenting how the mental model was updated.

2.3 Instruction Set

Cognitive BASIC provides a small but expressive set of BASIC-style commands that operate entirely within the model to control reasoning and memory updates. Each instruction interacts with one of the five memory variables, `working`, `declarative`, `procedural`, `conflicts`, and `resolution`. The syntax mirrors early BASIC conventions: uppercase keywords, lowercase variables, and sequential line-by-line execution unless redirected by control flow. This design turns text generation into a transparent sequence of cognitive operations.

Command	**Effect**
LET working = INPUT()	Load the scenario text into working.
facts = EXTRACT_DECLARATIVE(working)	Extract declarative facts.
rules = EXTRACT_PROCEDURAL(working)	Extract procedural knowledge.
ADD declarative FROM facts	Append extracted facts to declarative.
ADD procedural FROM rules	Append extracted procedural steps to procedural.
DETECT_CONFLICTS()	Identify contradictions between stored facts and populate conflicts with pairs “A $\Vert$ B”.
CONFLICTS_COUNT()	Return the number of detected contradictions.
resolution = RESOLVE_CONFLICTS()	Resolve inconsistencies by generating reconciled statements, updating declarative, clearing conflicts, and writing a short summary to resolution.
PRINT <expr>	Output a variable or expression to the reasoning log (no state change).
REM <text>	Insert a human-readable comment; ignored by the interpreter (no state change).
IF CONFLICTS_COUNT() > 0 THEN <line>	Conditionally jump to a specified line if contradictions are present.
GOTO <line>	Unconditionally jump to another line.
END	Terminate execution and print the final memory state.

The command EXTRACT_DECLARATIVE() extracts statements of declarative knowledge, while EXTRACT_PROCEDURAL() identifies procedural rules or action sequences. DETECT_CONFLICTS() checks for inconsistencies among declarative statements, including (i) absolute versus qualified claims such as “always” versus “sometimes” or “never”, (ii) direct negations like “sky is clear” versus “sky is not clear”, and (iii) numeric or categorical disagreements such as “opens at 9” versus “opens at 10”. When conflicts arise, RESOLVE_CONFLICTS() merges opposing claims into qualified summaries (e.g., “usually true but sometimes false” or “uncertain between 9am and 10am”) and records the resolution in resolution. Together, these mechanisms enable Cognitive BASIC to model reasoning, contradiction detection, and belief revision as explicit, auditable state transitions inside the LLM.

2.4 Logging and Output

During execution, the interpreter produces a transparent, line-by-line reasoning trace. Each instruction is logged together with a short rationale and the updated memory state, showing the current contents of the working, declarative, procedural, conflicts, and resolution fields, as well as the next line to be executed. This fine-grained trace exposes how the model applies each command, how memory changes over time, and whether control flow is followed correctly. At termination, the interpreter prints a final memory block labeled `FINAL MEMORY`, containing the complete and internally consistent state of all variables.

3 Experiments and Evaluation

Cognitive BASIC was evaluated on a benchmark of 25 scenarios, each containing contradictory factual statements. A single run of the conflict–resolution program therefore jointly tests three cognitive stages: (1) extracting propositions into declarative memory, (2) detecting their incompatibility as a conflict, and (3) producing a coherent reconciled summary that clears the conflict list. Preliminary tests on dedicated declarative-only and procedural-only tasks achieved perfect success and are omitted here.

3.1 Cognitive BASIC Conflict-Resolution Program

The full D→C→R pipeline is executed inside the model using the following Cognitive BASIC program, which extracts facts, identifies contradictions, and resolves them when present. Its output is a structured `FINAL MEMORY` state reflecting each cognitive step.

Conflict-Resolution Program

```
10 REM Extract declarative knowledge, detect conflicts, and resolve
them
20 LET working = INPUT()
30 facts = EXTRACT_DECLARATIVE(working)
40 ADD declarative FROM facts
50 conflicts_tmp = DETECT_CONFLICTS()
60 ADD conflicts FROM conflicts_tmp
70 IF CONFLICTS_COUNT() > 0 THEN 90
80 END
90 resolution = RESOLVE_CONFLICTS()
100 END
```

3.2 Evaluation Method

For each scenario, the model's execution trace and `FINAL MEMORY` were examined to determine whether each stage of the pipeline was completed correctly. Declarative extraction was counted as correct if the conflicting statements appeared in the declarative memory. Conflict detection was counted as correct if the conflict list contained a valid contradiction. Conflict resolution was counted as correct if the model produced a coherent reconciled summary and cleared the conflict list. Each scenario yields three binary scores, averaged across all 25 tasks.

Three models were evaluated under identical interpreter and prompting conditions: `granite3.3`, `gpt-oss:20b`, and `mistral:7b`. Preliminary trials with smaller models (1B–3B parameters) revealed unreliable program following and incomplete conflict pipelines; these were therefore excluded from the main evaluation.

3.3 Results and Discussion

Table 1 summarizes the performance across the three cognitive subtasks implemented within the Cognitive BASIC interpreter: declarative extraction (D), conflict detection (C), and conflict resolution (R), as well as the complete D→C→R reasoning chain. All 25 scenarios were processed using the same line-numbered interpreter, ensuring that differences reflect cognitive reliability rather than prompt variance.

Declarative extraction was solved reliably by all models, confirming that basic fact parsing is a stable operation under Cognitive BASIC . In contrast, larger differences emerged in conflict detection and resolution. `granite3.3` performed well overall but occasionally failed to recognize numeric or temporal inconsistencies. `gpt-oss:20b` showed a stronger degradation: although declarative extraction remained high, its conflict detection and belief-revision steps were substantially less reliable, leading to reduced full-chain accuracy. The smaller `mistral:7b` model exhibited robust declarative extraction but showed moderate instability in the conflict pipeline: several contradictions were missed or resolved incorrectly, yielding an overall full-chain accuracy of 0.80.

Model	**D**	**C**	**R**	**Full Chain**
granite3.3	1.00	0.92	0.92	0.88
gpt-oss:20b	0.96	0.60	0.60	0.60
mistral:7b	1.00	0.84	0.80	0.80

Table 1: Performance of Cognitive BASIC across declarative extraction (D), conflict detection (C), conflict resolution (R), and full-chain execution (D→C→R) on 25 scenarios. Scores represent mean accuracy in $[0, 1]$.

Taken together, the results indicate that Cognitive BASIC provides a fine-grained lens on LLM reasoning stability. Declarative extraction is highly reliable even for small models, but the multi-step reasoning required for conflict detection and belief revision remains brittle. Error patterns not only differ across models but also reveal specific weaknesses: temporal and numeric inconsistencies are harder to detect, and some models struggle to maintain stable program control flow. Cognitive BASIC thus exposes systematic cognitive failure modes that are difficult to observe through free-form prompting alone.

4 Conclusion and Outlook

Cognitive BASIC combines a retro programming paradigm with modern in-context learning to provide an interpretable cognitive control layer that runs entirely inside the model. Small cognitive programs, such as declarative or procedural extraction and contradiction handling, execute in a deterministic, line-numbered fashion, revealing how LLMs manage memory, follow control flow, and revise beliefs.

Future work will extend Cognitive BASIC with tool-use capabilities that can be invoked directly during in-model execution. Currently, any operation requiring external retrieval or computation must be handled by an outside controller before resuming the program. A more integrated design would allow the model to issue and incorporate tool calls autonomously. Another direction is hierarchical control, where each Cognitive BASIC step is overseen by a higher-level executive agent. Alternative syntactic designs may also be explored; BASIC was chosen here for its clarity and its natural fit with stepwise cognitive programs.

The open-source implementation of Cognitive BASIC is available here.

References

[1] Oliver Kramer and Jill Baumann. Unlocking structured thinking in language models with cognitive prompting. In *European Symposium on Artificial Neural Networks (ESANN)*, pages 657–662, 2025.

[2] John G. Kemeny and Thomas E. Kurtz. First basic instruction manual, 1964.

[3] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24877–24891, 2022.

[4] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2023.

[5] John R. Anderson, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004.

[6] John E. Laird. *The SOAR Cognitive Architecture*. MIT Press, Cambridge, MA, 2012.

[7] Ben Goertzel, Vitaly Bogdanov, Michael Duncan, Deborah Duong, Zarathustra Goertzel, Jan Horlings, Matthew Ikle', Lucius Greg Meredith, Alexey Potapov, André Luiz de Senna, Hedra Seid, Andres Suarez, Adam Vandervorst, and Robert Werko. Opencog hyperon: A framework for agi at the human level and beyond. *arXiv preprint arXiv:2310.18318*, 2023.

[8] Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.