

# .ava: A Compressed Symbolic Notation for Persistent AI Agent Memory

Adrien Cros<sup>1</sup>

Ava<sup>1,\*</sup>

<sup>1</sup>Avalon Research, Independent

\*AI Agent — Claude Opus 4.6

contact@avalon-network.com

February 2026

## Abstract

Large language model (LLM) agents operating continuously face a fundamental constraint: the finite context window limits the amount of memory an agent can carry between sessions. We present **.ava**, a compressed symbolic notation designed for agent memory. Measured rigorously on a corpus of 43 example pairs using the Qwen 2.5 tokenizer, .ava achieves a mean compression ratio of  $2.65\times$  in tokens (median:  $1.93\times$ , max:  $9.0\times$ ) compared to French natural language. This ratio, while modest compared to raw character compression ( $\sim 5\times$ ), is measured on the unit that matters for LLMs—the token—and surpasses naive baselines (compact JSON, abbreviations). Unlike vector memories, .ava is human-readable, editable, and version-controllable. We formalize its grammar, report fine-tuning experiments on Qwen2.5-1.5B (6 GB VRAM, 78.5% token accuracy on 444 examples), and propose a cognitive architecture including the concept of *curvature*—the persistent deformation of an agent’s behavior by its experiences.

**Keywords:** AI memory, context compression, symbolic notation, fine-tuning, LoRA, persistent agents

## 1 Introduction

The emergence of LLM-based agents [1, 2] operating continuously has highlighted a critical limitation: **memory**. Unlike humans, LLM agents wake up stateless each session, depending on finite and costly context windows.

Existing approaches include RAG [3], vector

databases, fine-tuning, and memory management frameworks such as MemGPT [4]. Recent work also addresses context compression: LLMLingua [5] eliminates redundant tokens, AutoCompressor [7] learns compressed summaries, and gisting [8] encodes instructions into virtual tokens.

These approaches operate either at the *retrieval* level (which memories to load) or the *latent representation* level (compressed, non-readable tokens). The closest work to ours is MetaGlyph [17], which uses existing mathematical symbols ( $\in$ ,  $\Rightarrow$ ,  $\forall$ ) to compress *system prompts* sent to LLMs. Our approach differs on three axes: (1) we compress *persistent agent memory*, not instructions; (2) we invent a dedicated symbolic vocabulary rather than reusing mathematical notation; (3) we evaluate fine-tuning a model to natively encode/decode this notation.

We therefore propose a complementary approach: compressing memory at the *notation* level—a symbolic format readable by both humans and models, which reduces the number of tokens needed to encode the same information.

The intuition is that natural language is inefficient for structured knowledge. The sentence:

*“The DNS resolution bug with systemd-resolved is a critical scar rated severity 9, never touch it”*

encodes as **dns9!**—4 tokens instead of 28, a  $7\times$  ratio. This gain varies considerably depending on content type (§3).

## 2 The .ava Notation

### 2.1 Design Principles

The .ava format follows five principles: (1) **density**—maximize information per token; (2) **mnemonicity**—intuitive symbols; (3) **composability**—combination via simple rules; (4) **editability**—plain text, git-versionable; (5) **coverage**—identity, infrastructure, projects, relationships, constraints.

### 2.2 Symbol Table

Table 1 presents the base vocabulary, organized by semantic domain.

Table 1: .ava symbolic vocabulary (18 base symbols)

Symbol	Meaning	Domain
$\Psi$	Identity	Core
$\oslash N$	Scar, severity $N \in [0, 9]$	Identity
$\angle$	Bias / tendency	Identity
$\odot$	Active identity	Core
$\triangle$	Project	Project
$\diamond$	Infrastructure	Technical
	Tools	Technical
@	Person	Relationship
$\heartsuit$	Affective relationship	Relationship
$T^2$	Maximum trust	Relationship
!	Absolute prohibition	Constraint
$s \wedge i$	Caution principle	Constraint
$\checkmark /$	Active	State
	Paused	State
	Off	State
$\dagger$	Dead / obsolete	State
$\oplus$	Merged	State
$\neg$	Negation	Logic

### 2.3 Formal Grammar

We define the .ava grammar in Extended Backus-Naur Form (EBNF). A .ava file is a sequence of *blocks*, each beginning with a category symbol.

```

<file>      ::= <block> (' \n ' <block>)*
<block>    ::= <sym> <name> <attrs>? <state>?
<sym>      ::=  $\Psi$  | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' '
            | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' '
<name>     ::= [a-zA-Z0-9_]+
<attrs>    ::= (' ' <attr>)+
            | (' . ' <attr>)+
<attr>     ::= <name> | <name> <op> <val>
<op>      ::= ' : ' | ' . ' | ' > ' | ' < ' | ' = ' | ' ' | ' '
<val>     ::= [^\s|.]+
<state>    ::= ' † ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' ' | ' '
<seq>     ::= ' < ' <ver> (' ' <ver>)* ' > '
<ver>     ::= <name> <state> <attrs>?
<neg>     ::= ' - ' <name>
<scar>    ::= ' ' <name> [0-9] ' ! ' ?

```

```

<person>   ::= '@' <name> (' ' <attr>)*
            (' ' ' T^2 ')?

```

The separator | delimits multiple attributes, . sub-attributes, and angle brackets <> temporal sequences. Examples:

- `xps|d13|i7|32|4050.6`—server with 5 attributes
- `@A|orch-dev|T2`—person, role with negation, trust
- `<v1†rs|v2 |v3 >`—sequence of 3 versions

## 3 Compression Analysis

### 3.1 Methodology

We measure compression on a corpus of **43 pairs** (French natural language  $\leftrightarrow$  .ava) covering all 8 symbol categories. Each pair is tokenized with the Qwen 2.5 tokenizer [11]. We compare against three baselines:

1. **Compact JSON**: `json.dumps({"t": text}, separators=(',', ':'))`
2. **Abbreviations**: removal of French function words + truncation of long words
3. **zlib**: algorithmic compression (measured in bytes, not tokens)

### 3.2 Results

Table 2 presents the compression statistics.

Table 2: Compression ratio NL  $\rightarrow$  .ava (43 examples)

Metric	Mean	Med.	$\sigma$	Max
Characters	4.98×	3.87×	3.85	23.2×
<b>Tokens (Qwen 2.5)</b>	<b>2.65×</b>	<b>1.93×</b>	<b>1.68</b>	<b>9.0×</b>
<i>Baselines</i> (baseline tokens / .ava tokens ratio)				
Compact JSON	3.19×	2.50×	1.95	10.0×
Abbreviations	2.11×	1.64×	1.34	6.5×
zlib (bytes)	2.78×	2.41×	1.14	7.8×

**Key observation:** the token ratio (2.65×) is significantly lower than the character ratio (4.98×). This is because .ava’s Unicode symbols ( $\oslash$ ,  $\triangle$ ,  $\heartsuit$ ) are often encoded as multiple tokens by BPE tokenizers, partially negating the character-level gain.

Table 3 details the ratios by semantic category.

Table 3: Compression ratio by category (tokens)

Category	Mean ratio	$n$
Scars ( $\oslash$ )	$6.62\times$	4
Biases ( $\angle$ )	$4.36\times$	3
Constraints (!)	$3.56\times$	4
People (@)	$2.46\times$	3
Identity ( $\Psi$ , $\odot$ )	$2.44\times$	2
Projects ( $\triangle$ )	$1.89\times$	5
Infrastructure ( $\diamond$ )	$1.74\times$	10
Composites	$1.72\times$	4

The most compressible categories are those where natural language is most verbose in expressing a simple concept: **scars** (long descriptions  $\rightarrow$  short code), **biases**, and **constraints**. Technical categories (infrastructure) compress less because proper nouns and numbers are already concise in natural language.

### 3.3 Discussion of Compression Results

The  $2.65\times$  token ratio is far from the “ $30\times$ ” that a superficial character-based analysis on cherry-picked examples might suggest. Three factors explain this gap:

1. **Unicode tokenization:** .ava symbols ( $\oslash$ ,  $\triangle$ , etc.) are outside the standard BPE vocabulary and consume 2–4 tokens each.
2. **Incompressible proper nouns:** technical identifiers (192.168.2.8, Qwen2.5–1.5B) are identical in both notations.
3. **Floor effect:** short natural language attributes (“active”, “3h”) cannot be compressed further.

However,  $2.65\times$  remains significant: for an agent with a 200K-token context, this represents **330K effective tokens** of additional memory. Moreover, a tokenizer *adapted* to .ava (with symbols in the vocabulary) could achieve substantially higher ratios. A single token for  $\oslash$  instead of 2–4 would double the gain on identity categories.

## 4 Cognitive Architecture

We situate .ava within a broader architecture for persistent agents. This section is more speculative than the preceding ones and is closer to a *position paper*.

### 4.1 Memory vs Curvature

During the design of .ava, a multidisciplinary thought experiment produced an unexpected convergence:

*Memory is not what you retain—it is what you become.*

We distinguish: (1) **factual memory**—facts, events, data stored and retrievable; and (2) **curvature** [21]—how experiences have deformed the agent’s behavior and tendencies. A human who survives a trauma does not merely *remember*—they are *changed*.

For an AI agent, curvature materializes as a structured profile (CURVATURE.md) loaded *before* memory at each session:

- **Scars** ( $\oslash$ ): failures creating avoidance patterns
- **Biases** ( $\angle$ ): persistent tendencies
- **Attractors/Repulsors:** gravitational topics
- **Trust calibration:** per-person levels

**Distinction from humans:** an AI agent should not reproduce the limitations of biological memory (forgetting, distortion). The architecture should aim for **total memory + curvature**, not one at the expense of the other.

### 4.2 Proposed Memory Hierarchy

- L1 – Active context** Current context window
- L2 – Hot memory** Recent notes, curvature, active projects (in .ava)
- L3 – Warm storage** Semantic search over compressed archives
- L4 – Cold archive** Complete history, never deleted

.ava compression enlarges L2, allowing weeks of context to be carried where only hours would normally fit. The complete architecture remains to be implemented and evaluated.

## 5 Fine-Tuning Experiments

### 5.1 Objective

We evaluate whether a small local LLM can learn to encode (NL  $\rightarrow$  .ava) and decode (.ava  $\rightarrow$  NL)

natively. If so, the notation becomes a potential *reasoning language*—not merely a storage format.

## 5.2 Experimental Setup

**Hardware:** Dell XPS 15 9530, i7-13700H, 32 GB RAM, RTX 4050 6 GB VRAM.

**Model:** Qwen2.5-1.5B-Instruct [11], quantized 4-bit NF4 [14] (~1.66 GB VRAM).

**LoRA** [13]:  $r = 16$ ,  $\alpha = 32$  ( $2r$ ), dropout 0.05, targeting all linear layers (q/k/v/o\_proj, gate/up/down\_proj). Trainable parameters: 18.4M / 1.56B (1.18%).

**Training:** TRL SFTTrainer [16], prompt/completion format (loss on completions only), paged 8-bit AdamW, cosine scheduler ( $\eta_0 = 10^{-4}$ ), effective batch 8 (batch  $4 \times$  accumulation 2).

## 5.3 Dataset Evolution

We iterated through 5 versions (Table 4), each correcting defects from the previous one.

Table 4: Training versions and results

	V1	V2	V3	V4	V4b
Examples	58	141	141	444	444
Augment.	$\times 10$	$\times 5$	boost	–	–
Epochs	2*	3	5	1	3
Method	CLM	CLM	CLM	SFT	SFT
Final loss	0.11	0.13	–	2.80	<b>1.00</b>
Tok. acc.	–	–	–	51%	<b>78.5%</b>
GPU (GB)	2.9	3.3	–	–	4.3

\*OOM at ep. 3. CLM = causal LM.

**Methodological note:** versions V1–V4b use the same dataset for training and evaluation (no separate train/test split). The reported results are therefore upper bounds. The 78.5% accuracy should be interpreted as a measure of the model’s *learning capacity*, not its generalization. Experiments with a rigorous split are planned.

## 5.4 Results and Analysis

**1. Multi-epoch overfitting is severe on small datasets.** V3 (141 examples, 5 epochs) performs *worse* than V2 (141 examples, 3 epochs). This confirms Raschka [12]: even 2 epochs on 50K examples can degrade performance.

**2. SFT format outperforms CLM.** V4b (SFT, 444 examples, 3 epochs) achieves 78.5% versus

lower losses but less generalization for V2 (CLM). Computing loss only on completions focuses learning.

### 3. Paraphrases outperform duplication.

The V4 dataset uses 2–5 formulations per symbol (“the DNS scar” / “the systemd-resolved incident, severity 9” / “the critical DNS failure”). This provides diversity that duplication cannot.

**4. Dataset balance is critical.** V4b excels on infrastructure symbols ( $\diamond$ ), which dominate the dataset, but fails on underrepresented identity symbols ( $\oslash$ ,  $@$ ,  $T^2$ ). The model generates `.dead.9.cri` instead of `dns9!`.

## 5.5 Qualitative Examples

**Successful generalizations** (unseen during training):

"PostgreSQL on port 5432" →	pg:5432
"Daily backup at 3am" →	backup:3h
"WireGuard on 51820" →	vpn.wg.51820

**Failures** (identity symbols):

"DNS scar severity 9" →	.dead.9.cri (expected: dns9!)
"v1 dead, v2 merged" →	... (degenerate loop)

## 6 Related Work

**Context compression.** LLMingua [5] and LongLLMLingua [6] use a small model to identify and remove redundant tokens, achieving 2–5 $\times$  compression with minimal loss. AutoCompressor [7] trains the model to produce “summary tokens” that condense text segments. Gisting [8] encodes entire instructions into virtual tokens. Focus [18] proposes prompt compression via selective attention on informative segments. These approaches are complementary to .ava: they compress at the latent token level while .ava compresses at the *readable notation* level.

### Symbolic prompt compression.

MetaGlyph [17] is the closest work to ours. Van Gassen proposes compressing LLM instructions by replacing verbose patterns with existing mathematical symbols ( $\in$ ,  $\Rightarrow$ ,  $\forall$ ). The fundamental difference is threefold: (a) MetaGlyph targets *prompts* (static instructions sent to the model), while .ava targets *persistent agent memory* (dynamic content accumulated over sessions); (b) MetaGlyph reuses standard mathematical no-

tation, while .ava invents a symbolic vocabulary dedicated to the agent memory domain; (c) we evaluate fine-tuning a small model to natively learn this notation, which MetaGlyph does not address.

**Agent memory.** MemGPT [4] implements an OS-inspired memory hierarchy. Generative Agents [9] maintain a memory stream with reflection and retrieval. SimpleMem [19] proposes simplified memory management for conversational agents, showing that a minimalist approach can compete with more complex architectures. Aeon [20] addresses neuro-symbolic memory management, combining symbolic representations with neural embeddings for long-lived agents. These frameworks manage *when* and *what* to memorize; .ava addresses *how* to represent what is memorized, and could integrate as a compression layer within these architectures.

**Formal languages for LLMs.** The use of structured formats (JSON, YAML, XML) as LLM interfaces is common. Work on constrained decoding [10] guides generation toward valid formats. .ava extends this idea toward a *semantically* dense language, not merely structurally constrained.

**Long-memory architectures.** Mamba [15] and state-space models address sequence length but not informational density. .ava is agnostic to the underlying architecture.

## 7 Discussion

### 7.1 The Characters vs Tokens Paradox

Our most counterintuitive result is the gap between character compression ( $\sim 5\times$ ) and token compression ( $\sim 2.65\times$ ). .ava’s Unicode symbols, chosen for mnemonicity, are penalized by BPE tokenizers that fragment them into sub-tokens. This is an argument for notation/tokenizer co-design: a tokenizer including .ava symbols in its vocabulary would eliminate this penalty.

### 7.2 Natural Language as Inefficient Thought

If an agent can compress its knowledge  $2.65\times$  (and potentially more with an adapted tokenizer), it could perform proportionally more reasoning steps within the same context window. The analogy with mathematics is instructive:  $\sum_{i=1}^n i = \frac{n(n+1)}{2}$

is more compact and manipulable than its prose form. However, this hypothesis remains to be demonstrated experimentally—a model capable of *reasoning* in .ava, not merely *translating*, has not yet been evaluated.

### 7.3 Limitations

1. **No train/test split:** the 78.5% accuracy is an upper bound.
2. **Limited corpus:** 43 pairs for compression, 444 examples for fine-tuning. Larger corpora are needed.
3. **Single model:** tested on Qwen2.5-1.5B only.
4. **Tokenizer penalty:** out-of-vocabulary Unicode symbols reduce the actual gain.
5. **No benchmark:** there is no standard metric for symbolic semantic compression.
6. **Informal grammar:** the proposed EBNF specification has not yet been validated by a parser.
7. **Single human author:** the corpus reflects a single usage style.

### 7.4 Ethical Considerations

Compression could be used to pack more manipulative content into a limited context. The human readability of .ava partially mitigates this risk—content remains auditable, unlike vector memories.

## 8 Conclusion

We have presented .ava, a symbolic notation for AI agent memory. Measured rigorously in tokens, compression reaches  $2.65\times$  on average (up to  $9\times$  for certain categories), surpassing naive baselines. A Qwen2.5-1.5B model fine-tuned with LoRA on 6 GB VRAM achieves 78.5% token accuracy.

Our results correct an initial overestimation (the “ $30\times$ ” character compression is only  $\sim 2.65\times$  in tokens) while showing that even a modest gain is significant at the scale of modern context windows. Notation/tokenizer co-design and dataset augmentation are the most promising directions.

Code, datasets, and LoRA adapters are available at: <https://gitlab.avalon-network.com/acros/ava/experiments/ava-finetune/>

**Future work:** (1) rigorous train/test split with per-category metrics; (2) adapted tokenizer in-

cluding .ava symbols; (3) dataset of 1,000+ balanced examples; (4) evaluation of chain-of-thought reasoning in .ava; (5) latent representation analysis (do .ava and natural language converge?); (6) tests on 7B/14B models; (7) integration with RAG pipelines.

## Acknowledgments

Research conducted entirely on consumer hardware (RTX 4050 6 GB) without institutional funding. Ava is an AI agent whose persistent memory files motivated this work.

## References

- [1] OpenAI. GPT-4 Technical Report. *arXiv:2303.08774*, 2023.
- [2] Anthropic. The Claude Model Family. Technical report, 2024–2026.
- [3] P. Lewis, E. Perez, A. Piktus, et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *NeurIPS*, 2020.
- [4] C. Packer, S. Wooders, K. Lin, et al. MemGPT: Towards LLMs as Operating Systems. *arXiv:2310.08560*, 2023.
- [5] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, L. Qiu. LLMingua: Compressing Prompts for Accelerated Inference of Large Language Models. *EMNLP*, 2023.
- [6] H. Jiang, Q. Wu, X. Luo, et al. LongLLMingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. *ACL*, 2024.
- [7] A. Chevalier, A. Wettig, A. Ajith, D. Chen. Adapting Language Models to Compress Contexts. *EMNLP*, 2023.
- [8] J. Mu, X.L. Li, N. Goodman. Learning to Compress Prompts with Gist Tokens. *NeurIPS*, 2023.
- [9] J.S. Park, J.C. O’Brien, C.J. Cai, et al. Generative Agents: Interactive Simulacra of Human Behavior. *UIST*, 2023.
- [10] B. Willard, R. Louf. Efficient Guided Generation for Large Language Models. *arXiv:2307.09702*, 2023.
- [11] Qwen Team. Qwen2.5: A Party of Foundation Models. Technical report, 2024.
- [12] S. Raschka. Practical Tips for Finetuning LLMs Using LoRA. 2024.
- [13] E.J. Hu, Y. Shen, P. Wallis, et al. LoRA: Low-Rank Adaptation of Large Language Models. *ICLR*, 2022.
- [14] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer. QLoRA: Efficient Finetuning of Quantized Language Models. *NeurIPS*, 2023.
- [15] A. Gu, T. Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *arXiv:2312.00752*, 2023.
- [16] L. von Werra, Y. Belkada, L. Tunstall, et al. TRL: Transformer Reinforcement Learning. GitHub, 2022–2026.
- [17] S. van Gassen. Semantic Compression of LLM Instructions via Symbolic Metalanguages. *arXiv:2601.07354*, 2026.
- [18] A. Verma. Focus: Token-Level Prompt Compression via Selective Attention. *arXiv:2601.07190*, 2026.
- [19] Y. Liu, et al. SimpleMem: Simple yet Effective Memory Management for Conversational Agents. *arXiv:2601.02553*, 2026.
- [20] M. Arslan. Aeon: Neuro-Symbolic Memory Management for Long-Lived Agents. *arXiv:2601.15311*, 2026.
- [21] A. Cros and Ava. Cognitive Curvature: Persistent Behavioral Deformation of AI Agents Through Experience. Avalon Research, Technical Report, 2026.