Dyalog North America - 7 April 2025

Dyalog and AI

Brian Becker
APL Tools Architect
Dyalog LTD

Mark Wolfson
CTO
Buyers Intelligence Group

Agenda

Brian

- Documentation
- Explanation
- Generation
- Integration

Mark

- Business cases for looking at AI
- Security Concerns
- Recent Efforts

What is an LLM?

An LLM or Large Language Model is a type of artificial intelligence system that processes and generates text based on patterns it has learned from vast amounts of training data.

At a very high level, an LLM works like a sophisticated prediction system. It analyzes sequences of text it has seen during training and learns to predict what words or tokens should come next in a sequence. When you interact with an LLM, it's essentially predicting the most appropriate continuation based on your input and its training.

In Dyalog APL terms, you might conceptualize an LLM as a complex function that transforms input vectors of tokens into output vectors, where the transformation is determined by billions of numerical parameters arranged in neural network layers.

AI Landscape

- Explosive Growth
  - More powerful LLMs

Anthropic Claude 3.5/3.7
OpenAI o3 and o1
DeepSeek R1
Google Gemini
Meta Llama
xAI Grok

AI Landscape

- Explosive Growth
  - More powerful LLMs
  - Multimodality

AI systems that work with text, images, and audio together

AI Landscape

- Explosive Growth
  - More powerful LLMs
  - Multimodality
  - Anthropic Model Context Protocol (MCP)

MCP provides a standardized way to connect LLMs with the context they need

AI Landscape

- Explosive Growth
  - More powerful LLMs
  - Multimodality
  - Anthropic Model Context Protocol (MCP)
- It's not going to slow down
  - LLM Leaderboard

Challenges

- Interface complexity
- Ecosystem limitations
- Documentation gaps
- Limited resources
- Keeping apace

Two Questions

- What can we do to use AI to help our users?
- What can we do to help our users use AI?

If you put tomfoolery into a computer, nothing comes out of it but tomfoolery. But this tomfoolery, having passed through a very expensive machine, is somehow enobled and no-one dares criticize it.

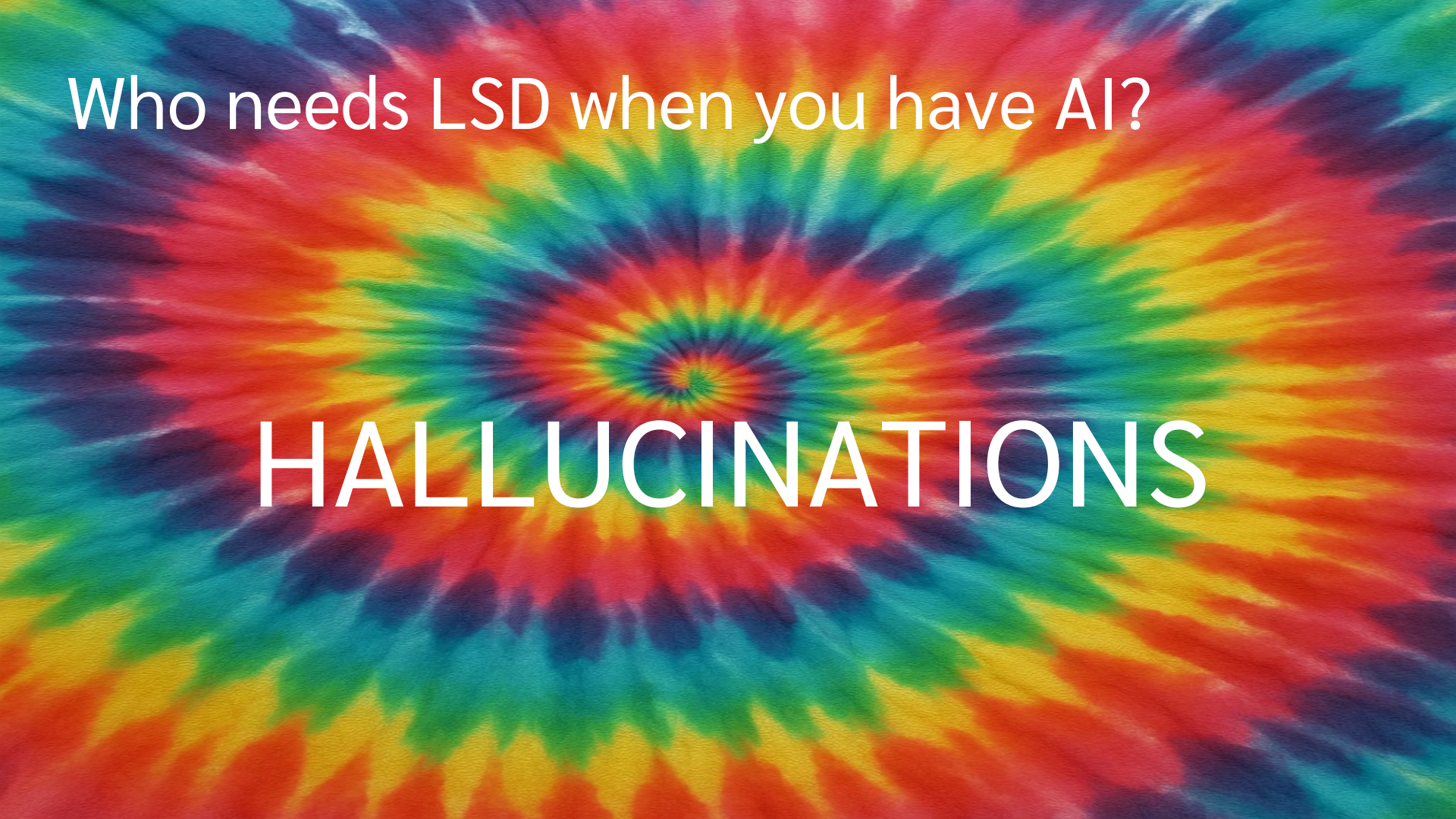
Pierre Marie Gallois

доверяй, но проверяй

Trust, but verify

Russian proverb popularized by Ronald Reagan

And sometimes, don't even trust, yet…


Who needs LSD when you have AI?
HALLUCINATIONS

BB What is Dyalog's Jarvis?

Dyalog's Jarvis is a specialized library designed for building graphical user interfaces (GUIs) in the APL programming language, specifically for Dyalog APL. Let me explain what makes it particularly interesting and significant.

At its core, Jarvis provides a way to create web-based user interfaces using HTML5, CSS, and JavaScript, but with the unique ability to control everything from APL. This is quite remarkable because APL is known for its array-oriented programming and mathematical notation, not typically for building modern web interfaces.

Think of Jarvis as a bridge between two worlds: the powerful numerical and array processing capabilities of APL on one side, and modern web technologies on the other. It's named after Tony Jarvis, a significant figure in the APL community who contributed greatly to making APL more accessible and practical for real-world applications.

Wh

Tell me about Dyalog's Jarvis

Dyalog's Jarvis is a web service framework built in Dyalog APL. Here's a breakdown of its key features and purpose:

- **Web Service Framework:**
  - Jarvis enables Dyalog APL developers to easily create web services. This allows APL functions to be accessed via standard web protocols.
  - It supports both JSON and RESTful APIs, making it versatile for various web application needs.
- **Purpose:**
  - The primary goal of Jarvis is to simplify the process of exposing APL code to the web or local networks.
  - It bridges the gap between APL's powerful computational capabilities and the accessibility of web technologies.

The sun will come out tomorrow

Annie

- Documentation
- Explanation
- Generation
- Integration

Documentation

- The current state
- mkdocs – v20.0 et al
- RAG – Retrieval-Augmented Generation
  - Notebooklm demo

Explanation

- "What does this do?"

Cursor

divvy2.aplf ●

C: > git > bb > divvy2.aplf

You, 2 months ago | 1 author (You)

```
1  divvy2←{(⍵/m)+(≠××)⌊+\⍵/t-m←⌊t←⍺÷⍵}
```

Understanding APL Function: divvy2

This is an APL function that appears to perform a division-related calculation. Let me explain it step by step:

apl — Apply to divvy2.aplf

```
divvy2←{(⍵/m)+(≠××)⌊+\⍵/t-m←⌊t←⍺÷⍵}
```

Breaking it down from inside out:

1. `⍺÷⍵` - Divides the left argument (⍺) by the right argument (⍵)
2. `t←` - Assigns this result to variable t
3. `⌊` - Floor function (rounds down to nearest integer)
4. `m←` - Assigns this floored value to variable m
5. `t-m` - Subtracts the floored value from the original division result
6. `⍵/` - Replicates based on the right argument
7. `+\` - Running sum (cumulative sum)
8. `≠××` - Sign comparison operation
9. The final result combines these operations

This function appears to be implementing a division algorithm that:

1. Takes a dividend (left argument) and divisor (right argument)
2. Performs the division
3. Handles both the integer and fractional parts of the result
4. Distributes the remainder across the result

For example, if you were dividing 10 by 3, instead of getting [3.333, 3.333, 3.333], this function would likely return something like [3, 3, 4] - ensuring the total adds up to the original number while distributing the remainder.

This is a fairly sophisticated APL implementation that handles division with remainder distribution. It's particularly useful in scenarios where you need to

@ divvy2.aplf

Ask, learn, brainstorm

Ask Auto Send

Claude

BB What does 2⊥⍣¯1⊢ do in APL?

In APL, `2⊥⍣¯1⊢` is an expression that converts a decimal number to its binary representation.

Breaking this down step by step:

- `⊢` is the right identity function, which simply returns its argument
- `⊥` is the decode (or base value) function
- `2⊥` specifically means "convert from base 2" (binary)
- `⍣¯1` is the inverse operator (negative power)
- So `2⊥⍣¯1` means "the inverse of decoding from base 2"

When you apply the inverse of base-2 decode, you're essentially performing base-2 encode, which converts a number to its binary representation.

For example, if you evaluated `2⊥⍣¯1⊢42` in APL:

1. The right identity function (`⊢`) would pass the value 42
2. The inverse base-2 decode (`2⊥⍣¯1`) would convert 42 to its binary representation
3. The result would be the array `1 0 1 0 1 0`, which is the binary representation of 42

This is a concise way to convert numbers to binary in APL, showcasing the language's focus on array operations and mathematical notation.

DYNA

Generation

- Produce some code that does _____.
- Write a test for _____.
- Python, JavaScript, HTML, CSS – piece of cake
- APL, not so much (yet)

Copilot

how do I compute the average of the columns of a table using APL?

In APL, you can compute the average of the columns of a table using simple array operations. Here's an example:

Apl Copy

```
avg ← ⌊+/÷≢
```

This expression calculates the average of each column in a table:

that's not correct.

You're absolutely right, and I appreciate your patience! To compute the average of the columns of a table in APL, here's the correct approach:

1. Suppose your table is a matrix named `table`, where each column represents a dataset.
2. To calculate the average for each column, you sum the elements of each column and divide by the number of rows.

The correct APL expression looks like this:

Apl Copy

```
avg ← (+⌿ table) ÷ ⍴⍴ table
```

```
avg ← (+⌿ table) ÷ ⍴⍴ table
```

- `+⌿ table` : Sums the elements of each column.
- `⍴⍴ table` : Retrieves the number of rows in the table.
- The division `÷` computes the average for each column.

Let me know if you'd like me to break this down further or provide more examples!

that's not right either - ⍴⍴ returns the number of dimensions in the table not the number of rows

You're absolutely correct again—thank you for pointing that out! I appreciate your sharp eye for detail. Here's the corrected APL code for computing the average of the columns in a table:

Apl Copy

```
avg ← (+⌿ table) ÷ ⍴ table
```

Here's how it works:

1. `+⌿ table` : Calculates the sum of each column (`⌿` operates along the first axis, summing each column of the table).
2. `⍴ table` : Gives the total number of rows in the table (not dimensions).
3. The division `÷` finds the average by dividing the column sums by the number of rows.

So now, each column's average is correctly calculated! Let me know if you'd like more help with APL or anything else you're working on.

that's not right either, ρ table returns the lengths of each of the table's dimensions

You're absolutely right again—thank you for catching that! I appreciate your persistence. Let's get the correct method for computing column averages in APL:

Given a matrix (table), you want to divide the sum of each column by the number of rows. Here's the proper APL expression:

Apl Copy

```
avg ← (+⌿ table) ÷ ⌈1⌷ρ table
```

```
avg ← (+⌿ table) ÷ ⌈1⌷⍴ table
```

Explanation:

1. `+⌿ table` : Sums each column of the table (`⌿` reduces along the rows).
2. `⌈1⌷⍴ table` : Retrieves the number of rows in the table. Here, `⍴ table` gives the shape of the matrix (e.g., rows and columns). `1⌷⍴ table` selects the first dimension (number of rows).

This will correctly compute the column-wise averages! Thanks for letting me refine my response—your questions help make it better. Let me know if you'd like to dive deeper into APL!

wouldn't +⌿÷≡⌿work just as well?

Yes, you're absolutely right— `+⌿÷≢` would work just as well and is a more succinct way to compute the average of the columns in an APL table.

Explanation:

- `+⌿` : Sums the elements of each column.
- `≢` : Returns the number of rows in the table.
- `÷` : Divides the column sums by the number of rows to compute the averages.

This is indeed the elegant and compact nature of APL at its best. Thank you for pointing out this cleaner alternative! Let me know if you have more APL insights or questions to share—it's always fun to explore together!

A couple days later…

how do I compute the column averages of a table in APL?

In APL, computing the column averages of a table is straightforward due to its powerful array-processing capabilities. Assuming you have a matrix (a 2D array) where each column represents a dataset, you can use the ⌸ (partition) and ÷ (divide) operators effectively.

Here's an example:

Apl Copy

```
colAverages ← (+⌸⍉table) ÷ ≢table
```

Explanation:

1. `⍉table` transposes the matrix so columns become rows.
2. `+⌸` computes the sum of each row (original column).
3. `≢table` gets the count of elements in each row.
4. Dividing the summed columns by their count yields the column averages.

Do you have a sample dataset you'd like me to help you work through? Let me know!

Generational Thoughts

- How useful is it?
  - to the APL Newcomer
  - to an experienced APLer
- But there's hope
  - More APL content to train LLMs
  - LLMs are improving, they handle mathematics fairly well and APL is arguably more syntactically consistent than math.
  - RAG – Retrieval Augmented Generation

Integration

- Tools and techniques for you to integrate AI
- OpenAI compatibility
- HttpCommand / REST APIs
- WebSockets for streaming
- Demo OpenAI and Anthropic

Some Practical Issues

- Where to apply resources?
- Who pays for it?
- Liability concerns

What can you do?

- Help us help you
  - Tools
  - Techniques
- Publish code

AI at BIG

AI that could be useful for BIG

- Generative AI using large language models
  - Used for creating content
  - Can create assistants that can be used for analysis of results
- Machine Learning
  - Learns patterns from examples to automate tasks
  - Can compare images to find similarities

Business use cases for BIG

- Optimizing model inventory per vendor
- Picking the correct products to order to restock from sales
- Automate reorders
- Based on attributes and images, find closest match products between vendors
- Redistribute inventory between stores (retailer)
- Redistribute inventory between stores (vendor)

Local AI or Commercial AI

- Privacy and security
- Scalability
- Resource considerations

Current Progress

- Local models installed on separate VM
- Using ChromeDB as vector database
- Rewriting Python utilities in Dyalog APL