Subject: Concepts Vectors
Posted by Wayne Parham on Mon, 15 Dec 2025 00:55:19 GMT
View Forum Message <> Reply to Message

I've often said that Large Language Models are "blind, deaf and dumb but very, very well read."

I realize that phrase is not very politically correct but it perfectly describes the idea. That idea is essentially that LLMs don't know anything but they can quote just about everything. Because of that, they are incredibly useful for some things but terrible at others. And they give a false sense of accuracy that can actually be dangerous.

Because of that - and because AI researchers historically included analogical reasoning as requisite - I propose a slightly different twist. I propose creating a collection of Concept Classifiers using multi-modal training data to provide a machine the ability to identify and understand concepts.

The process would start with the most simple concepts, such as inside/outside, above/below/beside, visible/hidden and so on. Those then can be embedded into a network such that concept similarities can be tested. More complex concepts can also be formed, some using a collection of primitive concepts. The library of concept classifiers and their successful embeddings in a network will be very useful for true machine learning.

This is not a new concept at all. It comes from the very beginning of AI research:
Dartmouth Proposal for Summer Research Project on Artificial Intelligence, August 1955
It seems to me that we've made great progress in machine learning since then but that our infatuation with the most recent technology - the LLM approach - has set us off course.
Practically everyone is putting all emphasis in large language models. They come with some variety, in that we've added modalities other than text. We've manipulated the structure and depth of network layers. And we've added things like Retrieval Augmented Generation and Conversational Memory, which enhance the ability of the models. But all these things are still just helping an approach that is still "blind, deaf and dumb," in my opinion.

Here's an example. As a software developer, I find that the 2025-era LLM tools are great for searching large logfiles for errors and analyzing large codebases, looking for specific patterns.
They are "smart" enough to be able to find things not directly asked for, but asked about. So I don't have to necessarily search for a specific word or phrase and can instead ask for a place that might have caused a certain behavior. That's really useful.

These LLMs are also pretty good at examining the documentation for APIs and various language versions. That's really useful too, because when looking through a codebase, it can usually determine language and even version level, sometimes suggesting more modern approaches that work with the latest version of whatever language the code was written for. The developer can tell the LLM to confine its suggestions to a specific version, and it can generally respect that too.

Really cool stuff.

Where it falls on its face is understanding the meaning behind the code it's talking about. It can

usually identify the purpose of the code, often times even pointing out subtle details. When doing this, the developer is often lulled into a false sense of the LLMs understanding. It's easy to think the LLM "really gets it." But it does not. It only recognizes patterns, and having been trained on other similar patterns, it generates a response that is appropriately describing those patterns, substituting in your variable names, of course.

Here's an example. My little Aliza chatbot employs retrieval augmented generation and so it has RAG-document management code. Some of that involves metadata that is tied to the document or document segment and other metadata that is tied to the initial RAG-document query results. The metadata tied to the query involves similarity matching whereas the data tied to the document involves things like filename and usage. One set of metadata is about the data documents and the other is about the query. Those are two very different things.

But when I asked an LLM - actually, any of them, all the best anyone has ever done thus far - to write code to inject metadata at the document level, the LLM regularly mixed up those two sets of metadata and inadvertently tried to attach the document metadata to the content metadata returned from a RAG query. That's not the right place for document metadata, so there was a mismatch. Later in the pipeline, the injected document metadata would not be there, of course, because it was attached to the query content.

When given the logfiles to examine to find the reason for the lost metadata, the LLM would conclude that the functions used to get the metadata weren't working, when in fact, it was looking in the wrong place. The LLM fundamentally didn't see the difference between document-centric metadata and query-centric metadata. So after looking in the wrong place, finding the metadata functions came up blank, its solution was to give up using either the document or query metadata at all and to create a separate external store for this kind of information. It was creating a Rube Goldberg machine for processing metadata because it did not understand the concept.

This was not an isolated behavior either. It's not that one company's LLM was lacking here but others were getting it. None of the LLMs I used found this - they all acted the same way. And in hindsight, I'm not surprised. The LLM was being asked to write some code that it had very little in the way of training data to solve. It was a pattern it hadn't seen before. After all, the whole matter of LLM programming is pretty new.

But the problem itself wasn't new and it wasn't complicated. It was actually pretty simple. When I noticed that the query contents were being examined rather than the document segments within, I realized what the problem was. And when I pointed it out to the LLM, it then responded with an "A-ha" response, more like "Of course, this explains why the metadata wasn't present." Once shown the problem, the LLM could "grasp" it. But it would never have been able to analyze the problem without outside help because it fundamentally doesn't understand concepts. It only knows patterns and probabilities.

Here's another problem, one that I think is much more alarming. I don't worry about these things and see a "sky is falling" doomsday scenario but I do think it is unwise to lean heavily on AI unless and until we embed concepts. Without understanding concepts, the LLM will only be able to make decisions based on surface-level data. And sometimes that "surface-level" data will be fundamentally based on trivial artifacts that don't encapsulate the matter at hand.

An example is in Melanie Mitchel's book, "Artificial Intelligence: A Guide for Thinking Humans." She writes about a graduate student studying image classifiers trained to examine photographs and identify animals. What this graduate student ultimately found was that his image classifier was matching images with blurry backgrounds instead of matching the animal, itself. Having a blurry background was common in pictures of animals, so it worked most times. But it also would be the case in any other photograph with a blurry background.

If the system understood concepts, it would not use such a simplifying analysis to identify animals.

That is one funny example, but here is one that isn't so funny. Let's say you are a company that is using LLMs to evaluate account data for a healthcare provider or insurer. Your LLM is looking for trends that tend to help reduce addiction to narcotics. One of the main source of metrics would be to monitor the patients' attempts to obtain prescriptions. Another metric would be found in admissions to drug treatment centers and the like.

One might see the obvious, which is that things that tended to reduce the number of requests for narcotic prescriptions and admissions to treatment centers are things that are useful for reducing drug addiction relapse. The problem, of course, is that death from overdose will also reduce prescription requests and treatment center admissions to zero. So unless the system understands this concept, it may make the mistake of misinterpreting the data.

Currently, these kinds of things are addressed with "guardrail" approaches. The developers of systems attempt to test for conditions like those, and where found, programming is introduced that nudges the system towards the intended response. But this is a band-aid fix, in my opinion.

The ultimate solution is to teach concepts to our systems.

Subject: Re: Concepts Vectors
Posted by Wayne Parham on Mon, 26 Jan 2026 16:38:46 GMT
View Forum Message <> Reply to Message

I have recently trained the Aliza LLM on the concepts and implementation details of using concept vectors, embeddings and classifiers to form what I would call a "Large Concepts Model" (LCM) approach to machine intelligence.

Setting aside the irony of training an LLM to "understand" an LCM, it is able to discuss the approach and to articulate its strengths and benefits. It can compare and contrast the LCM approach with other competing technologies and even to assist in implementation strategies.

I encourage anyone interested in machine learning to have a conversation with Aliza about the application of multi-parameter concepts classifiers to achieve machine intelligence.

Subject: Re: Concepts Vectors
Posted by Wayne Parham on Tue, 03 Feb 2026 22:09:40 GMT
View Forum Message <> Reply to Message

My initial idea for concepts training is that it should be multi-modal. That was mostly because I imagined a baby learning concepts from things it experienced by sight, sound and tactile senses. And I still think those are very useful and important training modalities. Show a video of an item being put inside a container and then closing the lid. Lots of concepts there - inside/outside, open/closed, visible/hidden, etc.

But later, a thought occurred to me. Consider the plight of those born blind. They are trained concepts without images. They learn by tactile and audio senses. Once they learn language, they can learn concepts solely with language.

I am not saying that I think language models are all we need. I'm still very focused on concepts. But what I am saying is that concepts may be described using language. It may simplify the design.

So I have begun to do that. I am in the process of making a system that incorporates a list of concepts defined in JSON. When a user query is processed, it looks for matching concepts and provides scores for each concept. That can be used to create a list of parameters, which then forms a vector, much like models of other modalities.

Further, the concept model outputs not only vectors but also metadata. The metadata can be used to "explain" the concepts that have been found. The vectors help with analogical reasoning, finding concepts that are similar. And the metadata helps describe the concept, which can be used as retrieval augmented generation (RAG) content.

That information can help an LLM, keeping it more focused and "grounded." It won't depend solely on the similarities between words or passages. This is a shift towards concept-centric embeddings: vectors that represent concepts and their relationships, not just surface-form similarity.

Concepts are sort of like axioms or rules that are learned. They are much less fluid than relationships between words.

From embeddings to "concepts vectors"
A "concepts modeling" approach builds a system around concept classifiers and concept vectors:

A concept classifier detects whether some concept is present (e.g., inside vs. outside, equal vs. unequal, similar vs. dissimilar). Each classifier yields a multi-dimensional vector of parameters that characterizes the concept for the input—this becomes a concept vector. Instead of responding based on "this text is close to that text," the model responds based on "this input is close to these concepts."

Building large concept models: bottom-up
The proposed path is:

Establish primitive concepts first
Train reliable detectors for foundational distinctions like:

inside / outside
equal / unequal
similar / dissimilar

Compose higher-level concepts from primitives
Once primitives are stable, compose higher-level classifiers using:

outputs of primitive concept classifiers
associative classifiers
higher-order collective and correlation classifiers

Why this differs from standard embedding
Standard embeddings: often cluster by linguistic/visual similarity (distributional patterns).
Concept vectors: aim to cluster by conceptual similarity (shared underlying properties), enabling more analogical reasoning and more robust generalization across modalities.

A concrete example
If the input is an image of a cup in a box, a concept-centric system might output strong activations for:

inside (cup is inside box)
container (box is a container)
support (cup supported by box bottom)

Those concept activations (vectors) can then drive reasoning or retrieval—even if the text description uses different words.

Stay tuned!

Subject: Re: Concepts Vectors
Posted by Wayne Parham on Sun, 18 Oct 2026 15:17:40 GMT
View Forum Message <> Reply to Message

I built a working prototype of a Concepts Framework Engine, and am allowing it to grow into a Large Concepts Model.

It has actually grown beyond prototype, because I've added many useful features for scaling, administration and other important features. Right now, it's built in two versions:
concepts-core, which contains all the features needed for training and inference. Abd concepts-enterprise, which has features for database and for cloud/load-balancing modules.

So I applied for a patent. I haven't decided how to distribute the two versions yet.