

(MoH)

MET[Ä]CADEMY OF HUMANITY
· (A) · {Ä} · [Ä] · {Ä} · (Ä) ·

DOCUMENT 7 · Research Essay · v1.0RC · 2026-08-17

AFTER VIBE CODING

How do we teach a machine not to lose the reason we started?

AFTER VIBE CODING

Imagine you finally decide to build a home. Not an architectural cube from a design magazine, the kind where nobody has ever fried an onion in the kitchen and sitting on the white sofa appears to violate international law. A real home. A place for morning coffee, friends, sex, arguments about those same coffee cups, books, work, children, old age, music at two in the morning, and the screwdriver that will be rediscovered eleven years later in a box of Christmas decorations. You give the future system your budget, the plot, the building regulations, the number of rooms, photographs of interiors you like. It calculates sunlight, wind, heat loss, plumbing, structural loads, insulation and produces a beautiful design. The fire-safety plan is so persuasive that any fire encountering the paperwork would quietly reconsider its life choices and burn somewhere else.

And it still might not be **your** home. The system knows how large a bedroom should be, but it does not know that for one person, at seventy-seven, a bedroom is where television, books, good light and quiet matter most, while for another it is one of the central spaces of intimacy and they would rather not wake up wedged between a wardrobe and the vacuum cleaner charger. It knows the optimal ratio between house and land, but not that you would happily trade twenty square metres of interior for a larger garden. It may place the study perfectly according to every architectural principle and still miss the fact that you work at night and want to watch first light arrive over the horizon. It may even know that you love having people over, without knowing whether that means two close friends beside the fireplace or fifteen people with guitars, wine and fifteen mutually incompatible theories about who among them can actually sing.

Every parameter can be satisfied while the central question remains untouched: **what kind of life is supposed to happen inside those parameters?**

That is roughly where programming is heading. Once, programmers explained almost every movement to the machine: take this value, put it there, compare these two things, repeat, continue. Then we learned to describe structures, objects, relationships and increasingly large pieces of a system at once. Today you can tell an AI, “Build me a booking service for a small hotel, elegant and simple, with payments, a calendar and a mobile version that does not look like a punishment,” go make coffee, and return to code, a database, an API and a button with a twelve-pixel border radius that nobody requested but which the machine apparently considers essential to civilization.

This is not merely a productivity trick. It is a historical change in the distance between **“I want this”** and **“here, it works.”** Things that recently required a small team can sometimes be built by one person in an evening. But fairy tales warned us about this problem long before computers existed. Magic is not most dangerous when it fails to grant a wish. The interesting trouble begins when it grants the wish **perfectly, but not in the way you meant it.**

You ask to become the richest person in town. The next morning, everyone else has mysteriously disappeared. Contract fulfilled. Software is usually less theatrical, but the mechanism can be similar. Somewhere between your intention, the AI’s interpretation, the technical architecture and the behaviour of the final system, something like `PAYMENT = AUTHORITY`, `UNKNOWN = FALSE`, `SIMILARITY = IDENTITY` or `CLIENT STATE = SERVER AUTHORITY` may quietly move in and start paying rent. An engineer already smells smoke. For everyone else, the translation is straightforward: the system may have decided that whoever pays automatically has the right to decide, that “we do not know” means “no”, that two similar people

or situations may safely be treated as identical, or that one participant's version of reality automatically counts as reality for everyone.

The code can be elegant. The types can be correct. The tests can all be green. The servers can be perfectly happy. Nothing is technically "broken". We have simply **implemented a small future apocalypse with excellent engineering discipline**.

This is where a question began to emerge inside IMAGO, a question we currently explore under the name **Meta.Semantic Programming**. It is not another language with another species of brackets. Humanity has survived enough brackets. The question is older and more human: can we carry not only an instruction, but also **the reason the instruction exists**? What matters here? What may change? What would be dangerous to change? Where do we know, and where are we merely assuming? What must remain true even when the form of the idea changes?

An idea rarely begins as a technical specification. At first it may barely have words. "I want a home." Then it unfolds. I want to see the sunset. I want a garden. I want friends to be able to stay overnight. I want work not to consume the bedroom. I do not want the old tree cut down for one more parking space. Eventually these become requirements, drawings, materials and construction. Only when you finally step inside does the question arrive that no spreadsheet can quite contain: **is this it?**

Software is beginning to travel through the same sequence, except the journey may now take ten minutes. A human intention becomes a prompt, the prompt becomes an interpretation, the interpretation becomes architecture, architecture becomes code, code becomes a system, and the system begins doing things in the world. At every transition, meaning can drift by only a few degrees. One degree here, two there, three at the next handoff, and eventually the ship arrives on the wrong continent even though every individual turn looked entirely reasonable. If we accelerate all those transitions by a factor of a hundred, asking **what exactly is travelling between them** stops being philosophical decoration and starts becoming engineering.

This is where MSL enters. The least misleading way to imagine it is not as another programming language, but as an attempt to give an idea a **passport for travel**. The passport does not merely say what should be done. It carries what must not be lost, which transformations are acceptable, where uncertainty remains, why a decision was made, who the result is for and what happened to the meaning when it changed form. If an idea has passed through five airports, we no longer care only that the green light says "luggage arrived". We also want to know whether the books, photographs and grandmother's letter arrived with it, or whether the suitcase now contains three socks and a microwave manual.

Current work around M4M pushes the question one step further. We usually imagine that an intention must survive translation into **code**. But what if, in some cases, the computational body itself should change? One task may demand clock-like determinism, another may benefit from controlled randomness, while a third may be better expressed as a form that develops through local interactions rather than being fully specified from above. In that world, the passport of meaning should not dictate whether the traveller flies, takes a train or crosses the sea. Its job is different: to help us determine whether, after all the transfers, the same traveller arrived, rather than a very polite stranger carrying the right suitcase.

And perhaps not every idea is a blueprint in the first place. Some ideas are closer to **seeds**. A blueprint tries to predefine the final form. A seed preserves origin, constraints, environment, relationships and rules of growth. You cannot specify the exact position of every future leaf, yet you can still tell an oak from a plastic palm tree. For MSL this changes the problem radically: sometimes what must be protected is not the finished shape of an intention, but its capacity **to remain itself while it grows**.

That is why RUN SUCCESSFUL can no longer be the final word. It tells us only that **something ran**. We are increasingly interested in another question: **did the thing that ran still correspond to what we meant?** An aircraft can execute its route flawlessly. There remains one small, occasionally emotional detail: was that the right airport?

And after arrival, we may want more than the result. We may want a **difference receipt**. Not a seven-hundred-page accounting novel, but a readable record: this was preserved; this was added; here the system inferred; here something was lost; here two meanings were accidentally merged; and here we are no longer certain whether the result is still the child of the original idea or was quietly switched in the maternity ward. Loss is not always an error. The real danger begins when a lossy transformation behaves as though nothing was lost.

At this point programming touches something humans have been doing forever. We have carried meaning through myths, laws, songs, mathematics, paintings, theatre, prayer, jokes, family stories, and the sentence "don't forget the bread", whose semantic weight changes dramatically depending on who said it and how many times you have already returned home without bread. Meaning has never lived only in words. It lives in relationships, context, expectation, memory and tone, in what is said and sometimes even more strongly in what remains unsaid.

This is why MSL is gradually moving away from the narrow idea of translating information and toward the broader problem of **migrating meaning between different bodies**. Text can become a diagram. A diagram can become code. Code can become

behaviour. Behaviour can become experience. Sometimes a new form can even reveal something the previous one could not express. Yet every transformation returns us to the same question: **what must this thing preserve in order to remain itself?**

There is another complication. Preserving every fact does not necessarily preserve the meaning. The same events told in another order may become another story. The same facts delivered in another sequence may lead a person toward another conclusion. So we increasingly have to examine not only information, but the **trajectory** through which information travels across a system and through a person. It is not enough to know which bricks were delivered. Sometimes we also need to know what kind of wall they were used to build, and on which side someone remembered to leave a door.

Now imagine a machine that does not merely say, “Done.” Imagine it saying, “Here is what I understood. Here is what I preserved. Here is where I made an assumption. Here is where some meaning may have been lost. I can implement this in three ways, but the third changes something you previously described as important. I think we should go back one step.”

That would not simply be a faster way to build software. It would be a different quality of relationship between intention and machine.

Some parts of human meaning may never be fully formalizable. That would not be a failure. If we discover the boundary beyond which a system must say, “I do not understand this well enough; a human needs to remain here,” that may be one of the most valuable results of the entire research programme. Intelligence does not mature when it acquires an answer to everything. It matures when it learns to distinguish knowledge from inference, and inference from mystery.

For decades we have asked computers one great question: **how do we teach you to do more?** AI has accelerated the answer dramatically. Meta.Semantic Programming begins with another question. It is slightly less impressive on an investor slide, but perhaps rather more important everywhere else: **how do we teach you not to lose why we were doing it in the first place?**