Meridian Research

Research Methodology

How I'm Actually Going to Kill Programming Languages

The Real Problem with Research

Most research fails because researchers spend 90% of their time on busywork and 10% on actual breakthroughs. They're updating websites, writing grant proposals, managing social media, tracking publications, and doing a hundred other tasks that have nothing to do with discovery.

I refuse to live like that.

Instead, I built a system where AI agents handle everything except the core research. While I'm thinking about natural language computing, my agents are maintaining our website, engaging with the community, tracking relevant papers, and managing all the operational stuff that usually kills research momentum.

This isn't about being lazy. It's about focus. When you're trying to revolutionize computing, you can't afford to waste time on administrative tasks.

My Agent Army (And Why It Works)

Here's what most people don't understand about AI agents: they get better when you give them real work to do. My agents aren't just running scripts. They're learning, adapting, and getting smarter every day.

Current agent roles:

- Website and brand management (because consistency matters)
- Research paper analysis and summarization
- Community engagement and feedback collection
- Documentation that stays up to date
- Performance monitoring across all our systems

But here's the key part: I create new agents whenever I hit a bottleneck. Need someone to analyze hardware specifications? New agent. Want better academic collaboration? New agent. Each one learns from the others and gets better over time.

The result? I spend 99% of my time on actual research instead of managing the research process.

The Four-Phase Attack Plan

Phase 1: Prove Natural Language Beats Code

What I'm building: Working prototypes that accept natural language commands and outperform traditional programming interfaces.

How I'll know it works: When developers choose natural language over code because it's faster, clearer, and less error prone.

This phase is about killing the biggest objection: "But programming languages are more precise!" I'm going to prove that's wrong. Natural language can be more precise than code when you build the right systems around it.

My benchmarks:

- 95%+ accuracy in understanding what users want
- Faster development time than equivalent coded solutions
- Fewer bugs and easier maintenance

Phase 2: Machines That Actually Talk to Each Other

This is where it gets interesting. Instead of rigid APIs and data formats, I'm building systems where machines coordinate through conversation.

What this looks like in practice: Your banking app doesn't send JSON to the server. It says, "John wants to transfer $500 to savings, but check his balance first and warn him if it's cutting things close." The server responds with questions, suggestions, or confirmations.

The technical challenge isn't making this work once. It's making it reliable, fast, and scalable. When machines are having thousands of conversations per second, every word matters.

Success means:

- 99.9% reliability in machine-to-machine communication
- Performance that scales with complexity
- Conversations that stay coherent even when they get complicated

Phase 3: Hardware That Thinks in Language

Current processors are amazing at math and terrible at meaning. That's the wrong optimization for a world where computation happens in natural language.

I aim to work with hardware partners to build processors optimized for semantic operations instead of arithmetic ones. Memory architectures that store relationships and context. Processing units that understand language patterns as efficiently as current chips handle numbers.

Target metrics:

- 10x faster natural language processing
- 5x better energy efficiency
- Seamless integration with existing systems

Phase 4: Making the Transition Actually Happen (Months 25-30)

The hardest part of any paradigm shift isn't proving it works. It's getting people to adopt it.

I'm building migration tools that make the transition from programming languages to natural language as painless as possible. Developer education that doesn't require forgetting everything you know. Community resources that answer the questions people have.

Success looks like:

- 10+ real systems successfully migrated from code to natural language
- 100+ developers who prefer natural language computing
- Tools that make the transition feel obvious instead of scary

How I Actually Do Research

The Learning Cycle That Never Stops

Every week, my agents compile what we've learned and what we should try next. Every month, I refine the methodology based on what's working. Every quarter, I step back and make sure we're still heading toward the right goal.

This is rapid iteration with continuous validation.

Hypothesis Generation on Autopilot

My agents, along with collecting data, generate new hypotheses based on what they're seeing. They design experiments to test promising ideas. They even run low-risk tests automatically and flag the interesting results for my attention.

It's like having a research team that never sleeps and never gets distracted.

Making Sure It's Actually Good

Testing Everything

Every system gets tested six ways from Sunday:

- Automated testing suites that catch bugs before I do
- Academic peer review through conferences and journals
- Real-world validation with actual companies
- Open-source releases so other researchers can verify our work

Benchmarks That Matter

Every system we build gets tested against:

- The best traditional programming approaches
- State-of-the-art natural language processing
- Industry-standard reliability and performance requirements
- Real user experience in actual work environments

Working with Others (Without Losing Focus)

Academic Collaboration

I aim to work with universities because they have smart people and interesting problems. I present at conferences because peer review makes everything better. I would like to publish papers because science works best when it's open.

But I don't let academic politics slow down the research. If a collaboration isn't productive, I end it. If a conference doesn't accept our work, I publish it elsewhere. The goal is progress, everything else is optional.

Industry Reality Checks

Technology companies keep me honest about what works in the real world. Hardware vendors help me understand what's possible with current manufacturing. Developer communities tell me what they need.

These partnerships matter because building something that only works in the lab is pointless.

Open Science When It Makes Sense

I share methodologies because better research methods help everyone. I publish datasets when it accelerates community progress. I contribute to open-source projects when they align with our goals.

But I'm not giving away the core breakthroughs that make this research commercially viable. Open science is great. Going broke while doing open science is not.

Resources and Reality (After funding)

Computing Infrastructure

I run high-performance computing clusters for my agents because they need serious computational power. I use cloud resources for large-scale experiments because scaling matters. I maintain hardware testing environments because you can't optimize what you can't measure.

This stuff is expensive, but it's essential. You can't revolutionize computing with a laptop and good intentions.

Working Solo (Mostly)

I work alone on core research because that's where breakthroughs happen. I collaborate with experts when I need specialized knowledge. I get mentorship from people who've done this before. I engage with communities when they can accelerate the work.

The key is knowing when collaboration helps and when it just slows things down.

What Could Go Wrong (And What I'm Doing About It)

Technical Risks

Natural language is ambiguous. Performance might not scale. The hardware might not work as planned. Legacy systems might be impossible to migrate.

For each risk, I'm running multiple parallel approaches. When one hits a dead end, I have alternatives ready. I validate continuously so problems get caught early. I consult with experts when I'm out of my depth.

Strategic Risks

The market might not be ready. Competitors might beat us to market. Key partnerships might fall through. The technology might work but nobody might care.

I'm managing these through early adopter validation, competitive analysis, partnership diversification, and IP protection. But honestly, some risks you just must accept when you're trying to change the world.

How I Know It's Working

Metrics That Actually Matter

- How fast we're generating and testing new ideas
- How much our natural language systems improve each month
- How many developers choose our tools over traditional programming
- How often our papers get accepted and cited
- How interested real companies are in partnering with us

Regular Reality Checks

My agents generate monthly progress reports automatically. Every quarter, I do a comprehensive review of what's working and what isn't. Every year, I bring in outside experts to validate that we're still on the right track.

If the metrics show we're heading in the wrong direction, we pivot. If the research isn't leading to breakthroughs, we change methods. If the market doesn't care about what we're building, we build something else.

Why This Approach Works

Most research fails because it's too slow, too bureaucratic, or too disconnected from reality. Academic research takes forever and often solves problems nobody has. Industry research moves fast but only tackles incremental improvements.

My methodology combines the rigor of academic research with the speed of industry development. Autonomous agents handle everything except the breakthrough thinking. Continuous validation ensures we're building something people want. Open collaboration accelerates progress without slowing down decision-making.

The result is research that moves at the speed of insight instead of the speed of bureaucracy.

The Bottom Line

Programming languages are going to die whether I kill them or someone else does. The question is who gets there first and builds the foundation for what comes next.

My methodology is designed to win that race. Maximum focus on breakthrough research. Minimum time wasted on everything else. Continuous validation that we're building the right thing. Rapid iteration until we get it right.

It's not traditional research. But traditional research isn't going to eliminate programming languages and rebuild computing from scratch.

This will.

This methodology isn't just how I do research. It's how I'm going to change the world.