Mary Grygleski

Monday, March 31, 2025 9:09 AM

@mgrygles
Linkedin: mary-grygleski
Github: mgrygles
Twitch: mgrygles

What is happening now is not really Artificial "Intelligence". It's more advance machine learning.

NLP: Natural Language Processing

Generative AI: Generates new content based on prompts based on machine learning and deep learning. Does "creative" stuff - non-deterministic.

GPT: Generative Pre-Trained Transformer

Ollama allows you to run the LLM on your local machine

https://github.com/zylon-ai/private-gpt
Lets you load your personal PDF files and then run queries against them

Prompt Engineering:
LLM - predicts the next token based on its training.

1) Write clear and specific instructions
    a. Use delimiters to clearly separate different parts of the input (instructions, examples, requested response format, etc.)
    b. Provide relevant details to perform the task
    c. Request structured output like HTML or JSON
    d. Include any conditions that should be verified before performing the task
    e. Provide examples of correctly completing the task (one-shot or few-shot prompting)
2) Ask the model to think through the response: "Let's think step by step"
    a. Chain-of-thought prompting
        i. Useful for task with multiple steps or tasks requiring complex reasoning
        ii. We include a "chain" or series of steps the LLM must complete before providing its answer

RAG

Combines Retrieval and Generative models

1. User submits a prompt to the RAG application
2. RAG Application retrieves relevant context from vector store
3. RAG Application sends prompt + relevant context to the LLM and receives a response
4. RAG Application sends response to user

Here's the link to the Spring AI RAG demo GitHub reference repo (by Dan Vega): https://github.com/danvega/java-rag/blob/main/README.md

Vector DB stuff
Tokenizing
Encoding
Embeddings

Agents and Agentic

- Agents
  - One who is authorized to act for/in the place of another
  - A computer application designed to automate certain tasks
- Agentic
  - *Agentic* refers to someone or something capable of achieving outcomes independently ("functioning like an agent") or possessing such ability, means, or power ("having agency")
  - It is especially used with a type of artificial Intelligence (AI), often referred to as an AI agent, designed to execute complex tasks autonomously or with little human involvement
- GenAI Agents (within the software context)
  - Software entities
    - Orchestrate complex workflows
    - Coordinate the activities of multiple agents
    - Process logic
    - Evaluate answers
- Multi-Agentic Systems
  - Multiple autonomous agents working together
  - Each agent specializes in different tasks
  - Emergent intelligence from agent interactions
    - Divide complex tasks into manageable subtasks
    - Leverage strengths of different AI models
    - Increase robustness and fault tolerance
- Fundamental Building Blocks
  - Within the Java context, and leveraging on Langchain4j's approach
    - AI Service
      - More flexible than the old "Chains" concept in Langchain
      - Declarative interface to the desired API underneath via an Object (proxy)
- Towards standardization(?) Model Context Protocol (MCP)
  - Originated by Anthropic
    - Open
    - Helps to build agents and complex workflows on top of LLMs
  - MCP Provides:
    - A growing list of pre-build integrations that your LLM can directly plug into
    - The flexibility to switch between LLM providers and vendors
    - Best practices for securing your data within your infrastructure

MCP Examples Github repo: https://github.com/modelcontextprotocol
Server examples: https://github.com/modelcontextprotocol/servers/tree/main/src
Someone wrote these thoughts on MCP: https://www.ondr.sh/blog/thoughts-on-mcp

Links to many links and tutorials can be found in the slides here:
2025-03-ArcOfAI-DemystifyingGenAI-Workshop - Google Slides

LLM GUI where you can play with stuff without having to go through a browser:
https://lmstudio.ai

Raj Keynote

Monday, March 31, 2025 7:24 PM

Wardley mapping:
Humans use Tools, Medium, and Language in order to reason about the world.

Imagine one of these three becoming the Google of tomorrow. (Google == portal to the Web)

No one controls all three (tools, medium, and language) until GenAI. If that becomes our portal to the world, it becomes our way to think about the world. Right now it is controlled by wealthy, amoral, corporations. Big problem.

What we need is Diversity, Critical Thinking, and Openness (Transparency)

Venkat Subramanium - Keynote

Monday, March 31, 2025 7:08 PM

It Ain't What You Think

AI is both fascinating and disturbing
Arthur C. Clarke -
Clarke's Laws:

1. When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he thinks something is impossible, he is almost certainly wrong.
2. The only way to discovering the limits of the possible is to venture a little way past them into the impossible
3. Any advanced technology is indistinguishable from magic.

Alan Turing:
Can machines think?
Turing Test: A computer could be said to 'think' if a human interrogator could not tell it apart, though conversation, from a human.

Almost every revolution is really an evolution for humans as it takes time to get traction.

Learn from history to determine how we should respond to this latest of revolutions.

Early 1900s - horses and carriages were everywhere. Unsanitary, unhealthy, expensive ($200/year), slow.
Automobiles ($35/yr. expense)
These shared the roads for nearly 25 years - because automobiles were not reliable.
Demand for carriage drivers went up. As well as demand for automobile drivers.
"Best thing since sliced bread" - Bread was invented 30,000 years ago
Slicer in 1912.
Took 25 years for it to become popular as they worked out the problems with it.

What is AI today?
AI -> Alternative Information
Absolutely Inconsistent

WE are responsible for AI's potential misinformation.

AI tools are fantastic at finding errors in code.
AI is an Awesome Investigator

What about code generation?
Writing code in functional style works very well.
Generating consistently high quality solutions doesn't happen.
Complex looking - possibly incorrect.
Sometimes doesn't compile.
Often solves the problem poorly.
AI -> Accelerated Inference

When I ask AI to do something I'm a novice at, I see it as awesome; but

When I ask it to do something I'm an expert at, I see it as awful.

AI is a tool. Leverage it, but don’t put all your faith into it.

AI has been trained on our own bad code.

Use AI to generate Ideas, not solutions.

I welcome strange ideas, but not strange solutions

AI is a black swan (a highly improbable event that, when it happens, changes everything)

It took several decades to create compilers. We did not become more trusting of compilers. Compilers became more performant and dependable - earning our trust.

Is AI the compiler of the future?

The one who is best ablet to adapt and adjust to a changing environment is most likely to succeed.

Skill up, don't fear down.

Architecture Patterns for AI-Powered Applications

Tuesday, April 1, 2025 9:08 AM

Michael Carducci
Book: Mastering Software Architecture

Machine learning has been on a long road, and only now people are able to see what progress has been made.

Is AI functionality or "illities" (potential issues that we have to consider as we move forward)

AI Illities:

- Cost
- Accuracy (hallucinations are a feature. It shows that there is an element of randomness that we depend on.)
- Security
- Privacy (You don't necessarily know where your data is going)
- Latency (We need target performance metrics. We can always make things faster, but that must be coordinated with the priorities of the business.)
- Throughput
- Observability
- Simplicity (overall complexity)
- Note: There are many more. These are the easy ones to think of.

First law of software architecture: Every decision in architecture is a trade-off.
Second law: If you think you've found something that isn't a trade-off, you just haven't found it yet.

Tradeoffs:

Accuracy vs. Cost
Accuracy vs. Privacy
Accuracy vs. Simplicity

Integrating AI Into Applications: The POC to Production Problem:

The "Guardrails" pattern is useful to ensure better security (protect against "prompt injection").
Have programmable guardrails for a good Gen AI solution

Context enhancement to personalize responses can be helpful, but they introduce latency and possibly privacy or security concerns.

Considerations of creating a fine-tuned model:

- Cost
- Data Management
- Model Management
- Performance
- Compliance & Governance
- Management at Scale

What if we host our LLM internally?

Security and Privacy concerns are improved
We have much more control over Latency and Throughput

Let's consider our tech stack to make this happen:

- Hardware (hard to source now, as well as the energy needs)
- Infrastructure
- Model (open source or closed source? General purpose or specialized?)

A Survey of Modern Tools:

- Vector Databases
- Search engines w/ semantic enhancements
- Cloud services for semantic search
- Pre-built frameworks and libraries
- Hybrid search technologies
- LLM APIs
- Enterprise platforms w/ semantic search

Video: Connecting GPT to Your Data with a Knowledge Graph

RDF: Resource Description Framework
Specification / mechanism for managing structured data

michael@magician.codes

Slides: https://magician.codes/assets/ai-patterns.pdf

Leveraging AI for Software Verification

Tuesday, April 1, 2025 10:46 AM

Peter West at Qualiti

Approaches to Software Quality
1. Testing - before release to production
2. Observability - post-release
3. Other: Data Quality, LLM Evaluations

Selenium is the current standard for UI testing. It can be flakey.
Most of the budget for testing goes to Web automation testing. And it's not very useful.
Today, it's outdated (Consider Playwright instead)

Codeless automation (e.g. Testim.io)
RelicX (map out the application and create a graph, or track user interactions and build out user flows - turning those into tests)

Now we're looking into GenAI to do this.

We need to test Intent rather than code validity

2010 - First run:
RNN - Recurrent Neural Network + GPT to create a sliding window to observe user interactions and train models on workflows.
Train LLMS to learn a "language" which is the application.
Then generate and run tests
Triage the output to make sure it matches what we expect
Maintain the tests
Problems
Are testing what we really need to?
Testing only the "happy path" - no problems or regressions
Testing was too low-level to satisfy stakeholders

Enter ChatGPT
GPT 3/3.5 was limited
GPT4 Finally intelligent. Smart enough to reason
GPT 4v/4o - vision and multi-modality. AI can see more than just the specific data you give it

New wave of AI: workflows and agents
- Prompt chaining: piping prompts and results into the next prompt in a chain of events
  https://anthropic.com/engineering/building-effective-agents
  Langchain, Langsmith are good tools for prompt chaining
  e.g. testing Expedia
  - Test Web site: 38 low quality tests
  - List features of Web site. Then generate tests for each feature: 238 quality tests

- Routing
- Parallelization
- Voting - Agents tend to vote in the same direction (may be based on prompts)
- Orchestrator-workers

- Numerical scoring (poor results. e.g.: GPT 4.0 always gives a 70)
- Evaluator-optimizer (moving towards agentic - depends a lot on context)
- Agents: LLMs using tools based on environmental feedback in a loop
    - Each agent is autonomous and not just copies of each other
- Collaborative
- Hierarchical - Manager/Supervisor

This is the Way

1. For Low Volume, mission critical actions
    a. Prompt chaining is best
2. For Important/High Risk, High or Low Volume Actions
    a. Again, prompt chaining works best
    b. Don't have addtional agents
    c. Continuously fine-tune your prompts
3. For low risk, but High Volume actions
    a. If complexity allows, minimize context while still separating steps
    b. Don't have additional agents
    c. Ensure your prompt is extremely well fine-tuned

Result 1: We must have Intelligent Tests: Eliminate noise, make it actually valuable
Result 2: Intelligent Execution: Act like a person running the test (expensive)

Qualiti Test Automation:

- Time to first test: 5 minutes! No learning curve.
- Missing a test? Ask for it?
- AI does all the work

Fine tuning your prompts will make a bigger difference than almost anything else.
Ask dumb questions, get dumb answers. Give not context, get generic answers.
Be overly clear like you would with an intern.

AI is expensive. Qualiti only uses AI to create tests. It uses Playwright for test execution.

Codium uses 7% to 50% of its costs on AI.

We don't need another chatbot

Guardrails:

- Constrained Access: if users can use your AI for something else, they will
- Filters: Use models with safeguards built in - you need to know jailbreaking and prompt injection has been addressed. Red-teamed only.
- Structured Output
- Human oversight: Collaborative intelligence
- (Avoid Deepseek)

Data Sensitivity

If you don’t give it data, it can't leak data. But it can't do much of use.
Palantir is now using OpenAI models via MS Azure on Dept. of Defense contracts. If they can, odds are it's good enough security for your use case.
99.999% chance that data leaks are going to come from your end, not theirs.
Focus on data security on your side, starting with SOC2.

The Jagged Frontier

AI is weird.

The context code wall - today.

1000 lines of code is near perfect

2000 starts to become unmanageable and undependable

Problems with Vibe Coding

1. Deletion of work - editing an existing file? If there are parts not related to the current request, they will quietly get removed (e.g. A search bar)
2. Make really bad architectural decisions and does stupid and dangerous things; e.g.: API keys in front-end, new DB tables, etc.)
3. Hand holding required - it might take a different approach if you're - try three or four different times. It doesn't know HOW shou want something done unless you tell it.

Test Driven Development

1. The biggest key to success on vibe coding real-world projects is testing
2. If you can't build your tests before you write your code, your requirements aren't clear enough
3. If you have tests prior to code, you give the AI something to build against

The 4 futures of AI

1. AI stagnates
2. Incremental Improvements
3. Exponential Improvements
4. AGI - AI is as good as, or better than humans at every task

Remember AI is an intern, and you don't trust interns with your customers.

Favorite tools:

1. ChatGPT & Claude
2. Perplexity
3. Langgraph, OpenAI Agents SDK
4. Zep
5. Ollama

Getting started with LLMS

Play with Claude
Play with ChatGPT
Follow Ethan Mollick - best AI thought leader

Navigating Software Quality: From Chaos to Control

Tuesday, April 1, 2025 1:00 PM

Vanya Seth - Head of Technology, ThoughtWorks India
@vanyaseth

Your future is a reflection of your past:

How has our industry changed over the years?

1. Agile Methods (~2001) - Iterative development, customer feedback
2. CI/CD (~2009) - Automated testing, frequent releases
3. DevOps (~2010) - Collaboration, shared responsibility ("shift left")
4. Microservices (~2013) - Independent deployment, scalability
5. Testing in Production (~2017) - Shift right testing, user feedback
6. ChatGPT (now) - AI as a commodity

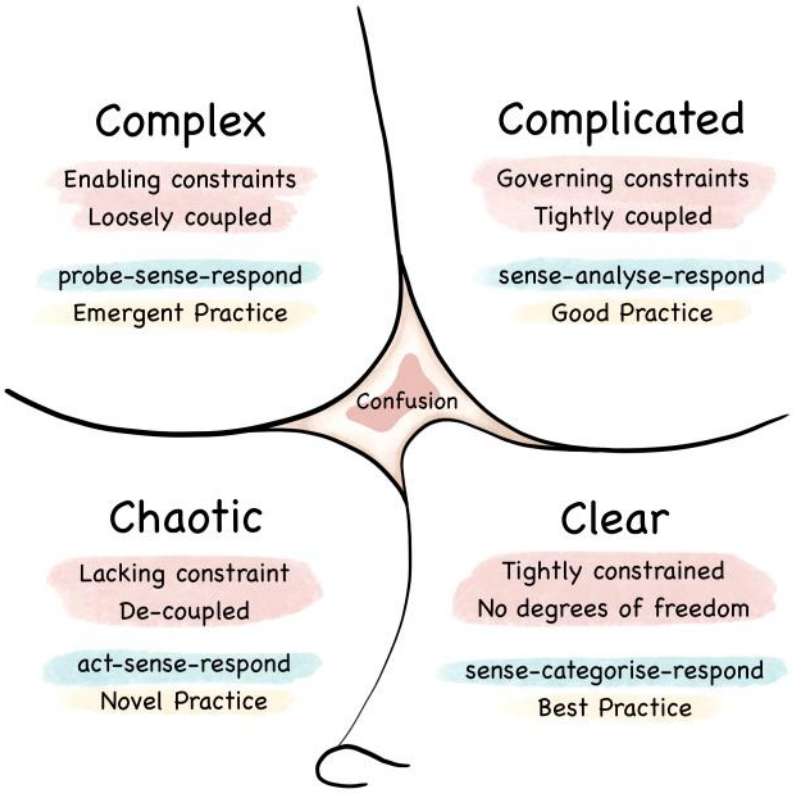
Hypothesis: Like it or not, All software will become AI/ML enhanced

The way we do our work is changing

The Cynefin Framework

Relates to organizational complexity
Decision making framework
Four domains

Clear	Known knowns
Complicated	Known unknowns
Complex	Unknown unknowns (Exaptive)
Chaotic	Unknowable / random



Testing: You expect a future, and then you verify that future

The Theory of Change

Vector moves in a certain direction.

The vector theory of change says that the destination isn't as important as the direction

Current direction of AI:
Stochastic -> Many correct and incorrect outputs
Subtle -> Incoherence & Hallucinations
Complex LLM chains
Evolving evaluation tenets -> Toxicity, bias, accuracy, relevancy
Evaluation required for both input and output
Sensitive information disclosure

Benchmarks aren't enough to validate an AI's usefulness and correctness.

AIs can be trained to optimize for benchmarks

RAG Triad:

When putting a question together, consider Context Recall and Context Precision
When evaluating the quality of an answer, consider the Relevance of the answer
When generating an answer from a context consider the

RAG vs. Prompt Stuffing

Tuesday, April 1, 2025 2:11 PM

Ken Kousen
Ken.kousen@kousenit.com
https://kousenit.org
Bluesky/Mastodon

LLMs are released fully trained (huge effort that we don't want to do)
How do we add information that:

- Is not on the public internet, and therefore is not available for training
- Wes created after the end of the training period

Three options:

1. Prompt stuffing
2. RAG
3. Fine tuning (you need many thousands of examples for more training.)

To do that, we put stuff into the context window.

Remember: Every prompt to the AI model is stateless. This is done by interactive systems by resending the previous conversation to the model.

Endo of training

- GTP-4o: Octoper 2023 (omni-model <==> multi-modal (video, image, sound, etc.)
- Clause 3.7: October 2024
- Gemini 2.0 and 2.5: Refuses to answer, but probably about November 2023
- Mistral Large: November 2024 - good at quick and dirty questions. Uses Flux 1.0 for image generation?

To find out, you can ask something like, "Who won the last two Super Bowls?"

We'll use Gemini 2.5.

Prompt stuffing

- Add the complete text to a message
- Needs to fit inside the *context window*

Context window sizes

- Gemini is 1-2 million tokens now. (Based on what you pay for)
- GPT-4o is 128K tokens
- Claude is 200K tokens
- Mistral is 131K tokens

Prompt Stuffing Pros:

- Works well for questions that require context
- Really easy to do -> no additional infrastructure needed
- Tokens (on some models) are very cheap
- Fast
- Good for questions that require multiple sections of data

Cons:

- Costs add up if you submit the same data repeatedly
- Limited by context window size
- Paying for unneeded tokens
- Questions about performance degradation as size increases

PDF Document loader: Apache Tika (also for MS Word docs)

Download info
- Document loaders
- jsoup library (old Java library for HTML parsing) https://jsoup.org
- Tavily (Web search engine designed to be called by AI tools/LLMs) https://tavily.com

RAG Frameworks for Java
Langchain4J (more advanced than SpringAI)
SpringAI (Should catch up in the next few months)

Langchain4J
- Java framework
- Competitor to Spring AI (both are good)
- "Universal" API for AI models
- Tool support / function calling
- Cool "ai service" support -> implements interfaces for you
- Manages chat memory
- RAG support
- Uses OK-HTTP as a Web client
- Has cacheing support to - which can save a lot of money
- Has a tokenizer for Gemini and OpenAI

Junit 5 Feature: @EnabledIfEnvironmentVariable() - Runs a test only if the environment variable is set

Estimating Tokens
- https://tiktokenizer.vercel.app
    - Generates tokens and encodings and counts tokens for any text

RAG - Retrieval Augmented Generation
- 2 stages
- First, populate and embedding store
- Second, execute query

RAG Pros
- Scales to much larger data sources
- Only relevant information included
- Updateable

Cons
- More complex
- Dependent on quality

RAG isn't necessary as much anymore with the larger token windows.
Sometimes RAG can lose the context that the Prompt Stuffing is able to keep

Managing Costs
Most LLMs provide support for prompt cacheing
- Reusing prompts

Prompt Injection - similar to SQL injection: inserting invalid information.

Example code is under the src/test/java in RAG package
https://github.com/kousen/LangChain4JDemo

Understanding RAG and RAG Techniques

Tuesday, April 1, 2025 4:01 PM

Pratik Patel @prpatel

Azul: Best JVM in the Galaxy!

Topics:
- Semantic Search Basics
- RAG & Embedding
- RAG Techniqus

Mistake-driven development

N8N - No-code flowchart/workflow tool

Ollama is a way to run LLMs locally
Qwen2.5:7b-8k - 8k token context

Searches: Keyword vs. Semantic
Keyword (lexical): Literal matches between the words in a user's query and the indexed content
Semantic :Understand the contextual meaning behind the words
- Relationships between works
- Previous search history
- Natural language processing
- Natural language understanding

Currently AI means Generative AI. There is also Predictive AI.
LLMs are a subcategory of Predictive analysis

Vector Embeddings
- Numerical representations of text
- High dimensional space
- Dividing large text/doc into smaller segments
- Used to then create vectors

Relationships
- Synonyms
- Antonyms
- Analogical

Similarity Metrics
- Cosine (angle/Direction)
- Dot Product (magnitude/direction) (not used as often)
- Euclidean distance

There are different embedding models used for different LLMs. They include more than just

How do we add to an existing LLMs knowledge?
- Fine tuning / LORA (layering a smaller neural network on top of an existing one)
- Agents

- Prompt injection / RAG (Retrieval Augmented Generation)

RAG
- Sounds complicated, but it's not - at a conceptual level
- Text is injected into the context of the LLM. That's it!

RAG Techniques
- Naïve RAG
- Contextual Retrieval

Chunking
- Have to split text into digestible pieces to create vector embeddings
- Different strategies
    - Chunking by character with overlap
    - Chunking by sentence

You can use context caching to keep from having to reparse a piece of text

When setting up a system to learn how this all works, remember that you don't need to create servers and database systems, etc. You can just keep everything in memory to test local tutorial data.

MCP - Model Context Protocol
- Pre-processes data that is fed into the context

Great tools
- Notebook LM from Google
- https://notebook.google.com
- Google Deep Research

What is AI Native Development

Tuesday, April 1, 2025 5:29 PM

Simon Maple, Founding Head of Developer Relations, Tessl

Snyk founder created Tessl https://tessl.io

Agenda
- Prompting with AI Assistants today to build software
- Prompting lessons learned
- AI Native development - a possible future?

Bolt.new

We're using words for similar things:
- Prompt
- Chat
- Spec

2 low-hanging fruits to get great improvements
- Prompt tuning
- Context tuning

When prompting won't help:
When the context needed to achieve the task is too large
- Some tasks may take several rounds of iterations
- Keep maximum context windows in mind

Sub-token level tasks
- LLMs work on the token level, so asking them for fine-grained character focused tasks is a bad idea
    - e.g.: code diffing, word manipulations

How LLMs read your prompt
- Tokens are processed sequentially, and predictions are dynamically

Anatomy of a prompt
Sandwich Technique
- Most important information goes at the start of your prompt, and is reinforced at the end

Formatting matters
- Structure your input - maybe like SGML or XML with sections and groupings
- Every LLM likes structure. The specific structure format doesn't necessarily matter.

Task Framing
- Often your "constraints" may actually be central to the task
    - e.g. "Do not use any external dependencies"
- Ideally, frame the task from the beginning rather than add a constraint at the end.
    - e.g. "Write a *self-contained JavaScript package*…"

Other tips
- Consider whether you're constraining the model by providing ***too much*** detail.
    - The more you constrain the problem the harder it will be for the LLM to satisfy your request.
- Model differences matter

- Chain of thought / homemade reasoning
    - Prime the model with context and THEN put in your request

Prompt Takeaways

- What can you implement right now?
    - Modularize and structure your prompts
    - Make sure your tasks are simple and well-scoped
    - Recognize the limitations of LLMs, and know when to implement a more technical approach

What is beyond prompts?

What if we compare the *code* and the *spec*?

LLM is magical, but unpredictable

Code couples what and app does and how it works

Software development is Code-Centric

- We start with a requirement
- From that we generate code
- Package that up
- Create tests
- Document it
- Now we need an enhancement
- We add some more code/tests
- Bug is reported
- Add more code/tests
- Edge case, more code
- Ecosystem changes, more code
- The list of all of the things that instigated a code change are lost.
- The docs become obsolete
- The code is the only thing that remains
- Even if it's no longer maintainable
- The changes to the code just accelerates over time. But the other stuff falls away.

THIS IS NOT THE **TRUE POTENTIAL** OF AI!

Transition from code centric to Specification Centric:

- In order to use AI, we can't make the code the center of truth anymore
- The Spec becomes
    - List of capabilities
    - List of requirements
- The code and docs can now be generated from the spec

LLMs let us decouple the **what** from the **how**

- What
    - Communicate in human natural ways
    - Language and text
    - Images and video
    - Voice and audio
- How
    - LLMS can write quality implementation code

LLMs fill in the gaps! It's what BDD promised us so many years ago!

Benefits of AI Native Development

- Software is less ***fragile***
  - To make this true, we need some sort of validation of the output
- Adaptable software
  - We need to provide the context of what is generated: language, OS, etc.
- Autonomous maintenance
  - Virtually every maintenance request that comes in, isn't necessarily a change to the specification. It would be just more information that goes into the context.
- Customizable software
- Dynamic software
- Personalized software
- Higher quality software
- Self optimizing software
- More inclusive space for developers

There are many questions still open:

- What's in a spec?
- How do you validate?
- How do you debug & observe?
- How do you engage with LLM decisions?
- How do you version adaptable implementations?
- What is the role of an AI Native developer?

AI Native is a new **development paradigm**

At Tessl, we're working on answers to these questions.

AI Native Development should be Open and Composable

- Who: Tool builders, experimenting users, opinionated developers, etc
- What: Share learnings,

https://landscape.ainativedev.io

What is Native AI

Four Pillars

1. From producer to manager
2. From implementation to intent
3. From deliver to discovery
4. From content to knowledge

https://ainativedev.io

News page describes this

From RAGs to Riches

Wednesday, April 2, 2025 9:03 AM

Heli Helskyaho

Generative AI without the context (ChatGPT):
Queries return lots of possible solutions rather than what you're looking for.

Hallucination and old data
We train the model with lots of old data
We then fine-tune it with newer or more specific data

When the LLM does not know the answer (does not have the data for the answer), it invents it (hallucinates)
This may be because the data is too old or that the data it used to train was erroneous
How do I check that the answer is correct?
Where did the model find the answer?

Generated text is non-factual and/or ungrounded. We don't know where it came from. No source.
It's often very hard to spot the errors.
This is probably the biggest problem with LLMs
There is no known methodology to guarantee the model is not hallucinating.
There are some methodologies to try and avoid hallucinations and to *ground* the response.

The model is not trained with my data. How do I add my data to the process?
Do I have to retrain the model with my data?
No, just use RAG.
"Augment" the prompt that you send to the LLM
RAG adds the context, augments the LLM with my data, adds it without retraining the LLM

Naïve RAG (Simple RAG) is simple and straightforward:
You ask a question
It vectorizes the question
It retrieves the basic info about the question
It augments the question sent to the LLM
This gives you a better quality answer

Vectors
Unstructured data (text, image, voice, video, etc.) is transformed (encoded, embedded) into numeric representation and stored as vectors.

Encoding <==> embedding <==> vectorizing

Choose an embedding model
Give it some data (sentences, etc.)
It will create vectors
The ones closer to one another are more similar

The LLM's job is simply to process natural language and generate text

Chunking: breaking a text source into smaller pieces

Naïve RAG is efficient when a single document contains the information needed for the response.
It has retrieval challenges:

- Getting relevant documents/data using only as simole similarity search
- Might pass irrelevant chunks or context to LLM or MISS rellevant data

Generation Problems

- Hallucinations, biases

Advanced RAG

- Multiple documents/sources are involved
- For complex tasks or open-ended questions
- Meant to improve retrieval accuracy
- Or reduce computational costs
- Or handle multi-modal data (images, audio, etc.)

Advanced RAG techniques

- Pre-Retrieval
    - Improve the quality of the data in the data stores
        - Data cleaning (remove unnecessary data)
        - Formatting
        - Organizing
        - Increase information density
    - Maybe use the LLM to create more dense data
    - Remove duplicated data
- Data pre-processing, indexing
- Retrieval process augmentation
    - Query expansion
        - We find relevant chunks for the query, modify it and pass along the modified query to the LLM
    - Context distillation
        - Summarizing/condensing retrieved documents to cover the most important information
        - Helps keep things less expensive and fit into the context window
        - Especially useful if the documents contain a lot of irrelevant content
        - Also helpful if the query involves complex or multi-step reasoning
    - Optimize search queries
        - Specialized LLM to take a question and put it into a format that is more easily understood by the next LLM in the query chain
        - https://www.falkordb.com/blog/advanced-rag - see Advanced RAG Techniques

Prompt Engineering

- Provide more context
- Structure queries for clarity
- Test different prompts/prompt formats
    - If the prompt is better, the response will be better

LongRAG

- Consists of a "long retriever" and a "long reader"
- Processes each individual document as a single (long) unit rather than chunking them into smaller units

https://arxiv.org/pdf/2406.15319
https://tiger-ai-lab.github.io/LongRAG
https://arxiv.org/pdf/2407.08223

https://arxiv.org/pdf/2408.00798 Golden Retriver
https://arxiv.org/pdf/2403.14403 Adaptive

Speculative RAG

- Smaller **specialist** model generates a **draft**

https://blog.google/produts/search/introducing-knowledge-graph-things-not

Htli.helskyaho@miracleoy.fi
@HeliFromFinland
https://Helifromfinland.com

Identifying and fixing Issues in Code using AI based tools

Wednesday, April 2, 2025 10:46 AM

Venkat Subramanium

Notes posted at https://agiledeveloper.com

AI Tools:

- Use different tools for different jobs
- Benefits:
  - Best at proposing ideas and finding issues in code
- Drawbacks
  - Its reporting may be overwhelming
  - You need to determine which is important and which can be ignored

Common errors that creep into code:

- Off by one
- Infinite loops
- Bad data
- Bad error handling
- Brace abuse
- Copy pasta
- Race conditions
- Concurrency problems
- Null Pointer Exceptions

AI is a low risk way to find high risk errors

Summary:
Don't use a tool. Use **multiple** tools.

Other tools:
Perplexity
Grok

Panel Discussion

Wednesday, April 2, 2025 10:52 AM

AGI - Artificial General Intelligence

- First we need to define it
- Regardless, we aren't close to having it working now
- Except: If the task is small enough we can have an AI perform it as well as a human can
- AI is trained on flakey Internet info. So are people!
- We don't have a singularity or artificial super-intelligence, but we do have very effective AI functionality.

Penning Powerful Prompts

Wednesday, April 2, 2025 2:17 PM

Craig Walls
@habuma
crag@habuma.com

"The show doesn't go on because it's read;
It goes on because it's 11:30"

Things are changing quickly and the material changes quickly, too!

Prompt: Essential question or instruction sent to an LLM

Prompts are expressed in natural language (which tends to be imprecise).
"A panda eats shoots and leaves"

LLM responses are based on statistics and random choices. These tend to be non-deterministic.

Mix an imprecise prompt and an non-deterministic response and … anything can happen!

The goal: Make prompts as precise as possible as that's the only thing we can control.

It's not much different from human-to-human interaction. We need to be specific and precise.

How do you write tests for something that is non-deterministic

1. LLM is a judge test. In SpringAI we use evaluators.
    a. What was your prompt?
    b. What did you expect to get?
    c. What did you get?
    d. What did a different model give?
2. Logs
    a. Log when the evaluator passes AND when it fails - including the WHY

Fundamental Techniques (YMMV - different models behave differently)

- Say what should be done, not what shouldn't be done
- Be very clear and specific
    - It's okay to have a prompt that is several pages long
- Offer structured input and ask for structured output
    - If you give your prompt in XML you can often get a better answer
    - Giving JSON, YAML, or a table, can be helpful
    - It's not good enough to get back a simple sentence as an answer
        - If you ask for JSON as a response, it will provide it.
- Assign a role
    - "You are a helpful assistant" helps!
    - "You are a helpful assistant who is a master of plumbing"
- Use the magic word: say "please"!
- Set prompt options
    - Not all models have these options
    - Starting with the phrase "I have a large collection of"

        - The model is consulted to pick the next token - which may be a small part of a word
        - It comes up with 5 candidates: books, coins, records, arts, stamps
        - Each of these tokens have a "logit" (a relative score of how confident the LLM thinks it should be the next token
        - Softmax turns logits into probabilities
        - The LLM then picks one randomly using a weighted choice. Pick a lottery ticket when some people have more than others. It may choose the lower probability.
        - "Temperature" adjusts probabilities to try and even the playing field a bit (2 = highest level, flattening the results the most. 0 = lowest level increasing the probability differences. This is sometimes done to make the results more deterministic. 0.7 is usually a good level)
        - Top-P - get the probabilities and sum up the ones that get to the Top-P point. (must be < 1)
        - Either use Top-P or Temperature, but not both
        - Top-K - pick the first N options based on highest probabilities.
        - Logit Bias - biases a lower probability option to have a better chance of being chosen. One option is more or less, the probabilities are recalculated and redistributed with the one option's bias taken into account.
            - Logit bias = 100 makes the biased option almost always selected.
            - Logit bias = -100 makes it almost impossible to select that option
        - These options are set usually in a JSON format in the request you send
        - SpringAI will handle all of this for you seamlessly.
- Prompting techniques (the design patterns of prompting)
    - See https://promptingguide.ai
    - Also read the prompting guides specific to the LLM provider you are using
    - One-Shot Prompting
        - Sending a prompt with some text and expecting an answer back.
        - Just getting straight to the point.
    - Few Shot
        - Give examples of the type of response desired
        - Great for giving guidance for how it might respond properly
            - Question: bla
            - Response: bla bla
            - Question: more bla
            - Response: bla bla bla
            - Question: Humina
            - Response:
    - Chain of Thought
        - Solving problems step-by-step
        - "Work through this step by step. The problem follows:"
        - "Okay, so here's the problem: "
    - Chain of Draft
        - Chain of thought with fewer details
        - https://arxiv.org/html/2502.18600v1
    - RAG (Retrieval Augmented Generation)
        - Lost its luster: Now people are focused on MCP
        - Retrieve some extra information
        - Augment the prompt
        - So we can generate a better response
    - Chat Memory
        - LLMs have no memory
- Agentic patterns
    - Per Andrew Ng

        - Reflection
        - Tool use
        - Planning
        - Multi-Agent Collaboration - one or more agents collaborating to come up with a good response
            - Until the critic is satisfied or the
        - https://youtu.be/KrRD7r7y7NY
    - Per Anthropic
        - https://www.anthropic.com/engineering/building-effective-agents
        - Prompt chaining

https://www.habuma.com
craig@habuma.com
SpringAI in Action - Manning
Spring in Action - Manning
Build Talking Apps for Alexa - Pragmatic

Conversational Analytics

Wednesday, April 2, 2025 3:56 PM

Sparql
Snowflake

Refactoring to Modernize Java Applications

Wednesday, April 2, 2025 4:02 PM

Venkat Subrmaniam
https://www.agiledeveloper.com - download link

Benefits
- Java is rolling out new features every six months
- This is one remarkable difference between Java and other languages
    - The way Java introduced Lambdas, is that you could use it with existing code
        - If any method had a functional interface, you could use it in a lambda expression
    - Other languages didn't have interoperability between old and new code

When you start a new job you look at three things
- Money
- What language do they use?
- What type of application are they building?
- How old is the code base? And is it poorly maintained?

Technical debt is not always caused by us neglecting something. Sometimes it's caused by time just moving forward. Languages evolve.

Challenges in moving forward:
- It costs time and money to modernize code. Hard to justify to management.
- If you're going to modify the code how much effort will it take to do so?
    - Refactoring used to be very expensive.
    - IDEs have made that much easier now
        - Some changes are still really expensive as they only work in small sections of code
- I know what modern features are available, but I don't remember every change that needs to be made to every aspect of the system to implement them properly. Use a tool to do this.
    - Even if you know what these changes are, the members of your team won't know.
    - What is the scale of the changes you need to make?
        - If thousands of applications need to be upgraded, it could be tremendously expensive

Tools can help us make the changes that need to be made
- Where are we now?
    - There are a few tools that can make some changes for you,
        - But they may not be able to do everything you need done
    - Maybe AI can help
    - Consider OpenRewrite

What does OpenRewrite do?
- Gives us hundreds of recipes that are already written (and being written) (Built-in & 3rd party)
- You can write your own recipes (custom recipes)
- AST - Abstract Syntax Tree
    - Parsed code
    - Details of syntax are abstracted out
    - You get the relationship of the elements of the code to one another
    - You DON'T get formatting details
- LST - Lossless Syntax Tree
    - Keeps every bit of information about formatting structure, comments, spacing, etc.

    - OpenRewrite uses this so it can preserve these details
- You can unleash it on your entire project.
    - Stores LST in memory
    - Larger codebase requires more memory
- IntelliJ has a plugin for OpenRewrite. It only works with IntelliJ's commercial version, not the community edition.
- OpenRewrite works with Maven projects and Gradle projects
- OpenRewrite is not for modernization. It is for refactoring
    - A subset of refactoring is modernization

How do you write a recipe

- Create a method matcher
    - Uses a method pattern to match a method
- Once it matches a method, it makes the required change

Example

- Before we start refactoring, what is the first thing we need to do? **TEST**
- Next we commit it to version control.
- Commit code frequently so that the cost of undo is zero
- `git restore` Done!
- ALWAYS run mvn rewrite:discover
    - If I made a mistake setting it up, *discover* will fail
- Bring in the recipe (e.g.: "UpgradeToJava21")
    - To do that properly, also include the "rewrite-migrate-java" recipe dependency
    - It gets the current version from the .pom file.
- Then mvn rewrite:run
- Git diff
- Mvn test
- Git commit

Using CoPilot

- Experimental Feature: Above your class, it will have a sparkle that says, "rewrite with modern Java"
    - It only worked for one day and then was removed.
    - CoPilot is using various internal tools
    - "Could you please modernize this code to a more recent version?"

Also look at Amazon Q (Cue?)

AI for Busy Java Developers

Wednesday, April 2, 2025 5:30 PM

Machine Learning Foundations
Patterns, Context, APIs, and Reality
Frank Greco
@frankgreco
@nyjavasig

Why Learn This Stuff?

- It's not just another framework, it's APIs
- https://github.com/langchain4j/
- We're just in the earliest days of the AI revolution
- There are patterns that we can recognize
- "The New Patterns of Innovation" by Rashik Parmer, Ian Mackenzie, David Cohn, and David Gann
- Before every revolution there is a period of instability
- Machine Learning is all about pattern recognition
- http://c2.com/doc/oopsla87.html
- Regex describes patterns in text
- 1943 - Artificial Neural Networks
- 1951 - Regular Expressions invented to analyze Artificial Neural Networks
-

You don't want to have to do the work to discover patterns in the data. Let the machine do it.

- Two models of machine learning
    - Predictive AI (been using this for 20 years)
        - https://www.jcp.org/en/jsr/detail?id=381 (Visual Recognition)
    - Generative AI (As of 2017)

We'll focus on Generative AI here

- Where there are patterns, there are opportunities!

Where are the patterns in software development

- Ideation
- Development
- Deployment
- Maintenance

Other patterns we live with

- Searching for info
- Problem solving
- Meetings
- Emails/texts
- Status reporting

Transformer Architecture Paper from Google: https://arxiv.org/pdf/1706.03762.pdf
Over 1.4 million models (just on Hugging Face!)
https://hugggingface.co/models

LLM is not really a search tool. It's really a text-pattern recognizer and probabilistic text-completion tool. It is not a fact-based search engine.

You can use it like that, but you will get burned.
You're talking to a collection of probabilities. You will get the most likely path, but it may introduce randomness, too. (Hallucinations)

You teach (train) it to recognize text patterns
You then ask it to complete

Your prompt guides the language model to provide a good answer. Bad prompt -> bad answer.
The context accompanying your prompt influences the quality of the answer.

AI Hallucinations are more correctly called "confabulations."

Prompt engineering is really prompt optimization techniques

- Zero Shot: Fire off a quick question to get a quick answer
- Few-Shot: You provide some context or background and ask for a response based on that
- Chain of Thought (CoT): Provide a series of question/answer pairs as a pattern to follow for the response you want
    - CoT is the basis of newer "reasoning" models
- Many others

Basic structure of a prompt

- Context
    - External info that can steer the model
- Examples (additional context)
- Input
    - Actual request
- Output indicator
    - Format or type of answer requested

Context influences the completion
Context can be added programmatically using a data store

Retrieval Augmented Generation <==> Embedding-enhanced Completion

When teaching, teach the **why** first, then teach them the **how**.

LangChain4J Components

- Services
- Core
- RAG

Fine-Tuned Model

- Enhanced model with special training
- You need to be an expert to do this properly

Determinism vs. Probability

- For probabilistic output, your testing has to change

If you want determinism, maybe you should use traditional automation

Know when to apply the right tool to the job.

Tools

- Cline.bot
- Cursor.com

Key Takeaways

Wednesday, April 2, 2025 9:53 PM

AI is still in its infancy, but it's changing EVERYTHING!

My job refactoring code will become irrelevant within the next 5-10 years.

In the meantime I need to learn the AI tools, and the ability to manage them, and the domains I need to know to use them to their best advantage.

People don't see what's coming. I can be among the heralds of the change in the industry.

Great tools to learn:

- Windsurf (using Gemini 2.5 or Claude 3.7)
- InstructLab

Get the slides for "Making LLM Fine Tuning Accessible"

Personal Projects:

- Learn about Machine Learning
- Learn how to set up a local LLM
- Learn how to create a RAG query
- Learn the various types of RAG queries
- Learn about MCP and how it can be used
- Learn about Agents and how they can be used
- Determine how I'm going to reinvent myself: going from Refactor Man to AI Maintenance Man
- Kcar.ai is available!

Work projects:

- Use AI to add a Web GUI to my app
- Set up a machine learning mechanism for the Work DB
    - Think about how to make it useful for the users
- Add a button/page/ui element to facilitate the users accessing AI analysis on their data via the legacy tool!

Keep in touch with Don Bogardus. He's my new muse!

Test Driving Code with the Help of AI

Thursday, April 3, 2025 9:05 AM

Venkat Subramaniam

TDD:

- Benefits of TDD
  - I do automated testing not because I have a lot of time, it is because I don't.
- If it's so good, why are we not doing it?
  - TDD Is a skill
  - We are busy chasing the cattle. We don't have time to fix the fence.
- AI to the rescue?
  - As of today, **no**.
  - We are talking about Test **first** development, not test **after**.
  - The interesting part of TDD is the **design**, not the tests.
    - I spend my time designing my code using tests
  - So AI does the part that I dread, and lets me to the interesting part.
    - I get to be creative in design. It does the mundane codeing.
- Letting AI drive
  - You need to be a smart navigator
    - The driver is only as good as the navigator

Different Tools:

- Copilot
- ChatGPG
- Gemini

Let's work with an example: Conway's Game of Life:

- https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
- First: Create a test class
  - If it's a well-known problem, AI should be able to create a test class quickly
    - Bad tests lead to bad design
    - A design is bad if it has unnecessary state on it
    - You don't want to write code where you set a state and get a state
  - **Lesson**: I want to drive the behavior of the code. Not state manipulation.
  - Start typing a test name and it will often figure out where you're going with it.
    - When you are clear where you want to go, the AI will follow your lead and start running in that direction
    - It will try to generate an entire function, but since we're doing TDD, we just want the minimum to make the test pass
      - // minimum code to make the test pass, please
      - If it still can't pick up on what you want, just write the minimum code
  - **Lessons**:
    - It does not know your design ideas
    - It takes you down the wrong path
    - Tug it towards where you want to go and it will figure out your intent
    - It has a starting problem
    - It learns from you, but quickly may write tests that ware actually incorrect
    - It has a turning problem - wants to take the design in a direction I don't want
    - It has a stopping problem - wants to keep retesting that issue

        - It often leaps forward, learning from you quickly
        - If it does something other than what you want, type over and it picks up real soon and goes along with you
        - It rapidly picks up the patterns in code from you
    - Design Rule:
        - Never put multiple asserts in a single test.
    - Create a test. Let it generate the code for it.
        - If the test fails, have it regenerate a few times.
        - If it still fails, write the code yourself.
        - Create a comment describing what you want: e.g.: please implement this using a ternary operator
    - If you don't know where to go for the next step, set a timer for 15 minutes to come up with some ideas.
        - If you're still stuck after that, ask the AI for a suggestion
    - Remember: It's your responsibility to create the software, not type the code
    - Keystroke to generate a different option: ESC

So: AI is a driver that matches the Navigator's intelligence

To learn better design skills, pair with better developers

A Practical Introduction to LangChain4J

Thursday, April 3, 2025 10:47 AM

LLMs
- Lots of them

Accessing them
- Useful as a user, but not for program

Accessing them programmatically
- Use as a tool within a product to make it good for my users
- What problem do we want to solve?
    - Want to talk to a MySQL database
        - Get MySQL classes
        - Client code talks to MySQL classes
        - But we want to use a different DB now
            - We don't want to change our code to support the new DB
            - This is "tight coupling"
        - Use Dependency Inversion Principle -> Use interfaces
        - We call this JDBC
        - JDBC is to databases as LangChain4J is to LLMs
        - Adapters are an implementation of an interface.
            - Allow you to connect your application to a something

Tight Coupling makes it hard to switch

Dependency Inversion Principles
- Client ---> Interface <|--- Adapter ---> Implementation

LangChain -> Python tool

LangChain4J -> Java implementation of LangChain
- Interface for various LLMs to facilitate decoupling programs from specific tools

Provides a set of interfaces and adapters
- Inference engine that facilitates queries

Let's use a model and switch

Keys to access the models
- HUGGING_FACE_API_KEY
- OPENAI_API_KEY
- Decouple our code from the library:
    - static var HUGGING_FACE_API_KEY = System.get("HUGGING_FACE_API_KEY");

Query about a person

This is a builder to create the connection interface tying me to a driver

```
ChatLanguageModel chatLanguageModel = HugingFaceChatModel.builder()
        .apiKey(Keys.HUGGING_FACE_API_KEY)
        .temperature(0.1)     // set the reality vs. creativity parameter
        .build();
```

System Message
- Describe the context of the system: "You are an experienced software developer and you want to help people to understand software development"

User Message
- Query from the user: "Tell me something about Brian Sletten"

RAG Ingestion
- Embedding your own data

    - Text splitter - split your texts into smaller, more manageable pieces
    - Segments - result of the splitter
    - Embedding models - format of the segments
    - Embeddings - formatted units that the RAG can process
    - Embedding store - database holding embeddings

RAG Retrieval

- Query
- Query embeddings
- Relevant segments
- Query + Relevant Segments

Various stores

Using an in-memory store

Computers Aren't Brains and Bots Aren't People

Thursday, April 3, 2025 1:01 PM

Brian Sletten (bsletten@mastodon.social
https://tinyurl.com/aa-2025-brains

Children's book: A Flower Pot is Not a Hat
Explores how one thing could be treated like something it wasn't originally intended to be.

What is AI?

- Is the intent to automate human labor? That's just the wrong goal.

We are more than just deductive systems.

Inductive learning: We observe trends and patterns, and extrapolate assumptions based on those trends.

Bertrand Russel's deductive turkey

Deductive reasoning only helps with data that we are exposed to.
I haven't died yet, so I assume I won't die.

We can make observations, but we can't be confident in the conclusions we draw based on these observations - as we haven't observed *everything*.

Adductive inference: our ability to come to a conclusion based on a number of different inputs

The brain uses a combination of inductive + deductive + adductive reasoning.

AI doesn't do that. And if your application doesn't need all three, that is fine, but don't try to call it AGI.

https://aibusiness.com/ml/why-the-turing-test-is-wrong

https://arcprize.org/blog/announcing-arg-agi-

https://www.health.harvard.edu/blog/right-brainleft-brain-right-2017082512222

https://bexleytorch.org/2021/03/10/myers-briggs-type-indicator-faces-modern-scrutiny

Presentation Topics

Friday, April 4, 2025 2:48 PM

"How Machines Learn: A Simple Guide to Machine Learning"
– A beginner-friendly breakdown of how ML models are trained and used, with fun examples.

"Prompt Engineering: The Art of Talking to AI"
– Teach people how to craft effective prompts for tools like ChatGPT or Midjourney.

"Ethics in AI: Bias, Privacy, and Responsibility"
– An important and thought-provoking topic about the social impact of AI.

"Using AI to Refactor Legacy Code: A Practical Guide"
– Show how LLMs like GPT can assist in understanding, cleaning, and modernizing old codebases.

"Building AI-Native Tools for Modernizing Technical Debt"
– Explore how to create tools that apply AI models to detect smells, suggest refactors, or generate tests.

Links

Saturday, April 5, 2025 3:07 PM

Arc of AI Conference: https://www.arcofai.com/schedule

Discussions from Arc of AI 2025 Workshop · ai-ml-workshops · Discussion #2: https://github.com/orgs/ai-ml-workshops/discussions/2

ai-ml-workshops/ai-quarkus-langchain4j-chatbot: https://github.com/ai-ml-workshops/ai-quarkus-langchain4j-chatbot

Vector Databases: A Technical Primer: https://tge-data-web.nyc3.digitaloceanspaces.com/docs/Vector%20Databases%20-%20A%20Technical%20Primer.pdf

ai-ml-workshops/vector-db-samples: A folder with samples on how to work with vector DB(s): https://github.com/ai-ml-workshops/vector-db-samples

Introduction :: Spring AI Reference: https://docs.spring.io/spring-ai/reference/

404 - Not Found - Quarkus: https://quarkus.io/guides/langchain4j/

Hugging Face – The AI community building the future.: https://huggingface.co/welcome

AI-ML-workshops (AI/ML Workshops Learning): https://huggingface.co/AI-ML-workshops

2025-03-ArcOfAI-DemystifyingGenAI-Workshop - Google Slides: https://docs.google.com/presentation/d/1ViDvh-_pyi7Dm_7IiTjtvfJJJLSomOTWF0evcvdg-wI/edit#slide=id.g324a9713850_0_0

should i install docker on windows or wsl - Search

Get started with Docker containers on WSL | Microsoft Learn: https://learn.microsoft.com/en-us/windows/wsl/tutorials/wsl-containers

WSL | Docker Docs: https://docs.docker.com/desktop/features/wsl/

Install WSL | Microsoft Learn: https://learn.microsoft.com/en-us/windows/wsl/install

spring-projects/spring-ai: An Application Framework for AI Engineering: https://github.com/spring-projects/spring-ai

mgrygles (Mary Grygleski): https://github.com/mgrygles

docs/market-report.pdf - Search

Introduction - Model Context Protocol: https://modelcontextprotocol.io/introduction

liveProject - premium training by Manning: https://liveproject.manning.com/module/1640_2_1/chatbot-with-llama/setup/libraries-and-setup

manning-lp/ScottBob-chatbot-with-llama-lp: My repository for liveProject: Chatbot with Llama:

https://github.com/manning-lp/ScottBob-chatbot-with-llama-lp

InstructLab: https://instructlab.ai/

wage stagnation in nine charts - Search
Wage Stagnation in Nine Charts | Economic Policy Institute: https://www.epi.org/publication/charting-wage-stagnation/

wardley mapping - Search

ArcOfAI - Leveraging AI for Software Verification - Google Slides

in-memory vector database - Search
An Honest Comparison of Open Source Vector Databases - KDnuggets: https://www.kdnuggets.com/an-honest-comparison-of-open-source-vector-databases

Cursor - The AI Code Editor: https://www.cursor.com/

Qualiti - an AI-managed Test Automation Solution: https://www.qualiti.ai/

Which AI to Use Now: An Updated Opinionated Guide (Updated Again 2/15): https://www.oneusefulthing.org/p/which-ai-to-use-now-an-updated-opinionated

carducci (Michael Carducci): https://github.com/carducci

kousen/LangChain4JDemo: Project with LangChain4j examples, some of which are from the langchain4j-examples project on GitHub.: https://github.com/kousen/LangChain4JDemo

Powerful Workflow Automation Software & Tools - n8n: https://n8n.io/

AI Native Developer Tools Landscape: https://landscape.ainativedev.io/

Code written by AI, powered by your specs | Tessl.io: https://www.tessl.io/

News | AI Native Dev: https://ainativedev.io/news

agi ai - Search

ChatGPT | OpenAI: https://openai.com/chatgpt/overview/

cfjedimaster/gemini-ai-preso: https://github.com/cfjedimaster/gemini-ai-preso/

java machine learning tutorial - Search

Java Machine Learning: https://machinelearningmastery.com/java-machine-learning/

Prompt Engineering Guide | Prompt Engineering Guide: https://www.promptingguide.ai/

Chain of Draft: Thinking Faster by Writing Less: https://arxiv.org/html/2502.18600v1

Andrew Ng Explores The Rise Of AI Agents And Agentic Reasoning | BUILD 2024 Keynote

Downloads and Resources - Agile Developer: https://agiledeveloper.com/web/downloads

OpenRewrite by Moderne | Large Scale Automated Refactoring | OpenRewrite Docs: https://docs.openrewrite.org/

Welcome to Moderne | Moderne Docs: https://docs.moderne.io/

The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 381: https://www.jcp.org/en/jsr/detail?id=381

predictive AI - Search

What Is Predictive AI? | IBM: https://www.ibm.com/think/topics/predictive-ai

Why the Turing Test is Wrong: https://aibusiness.com/ml/why-the-turing-test-is-wrong#close-modal

AI Native DevCon - 2025 Spring Edition: https://ai-native-devcon-2025.heysummit.com/