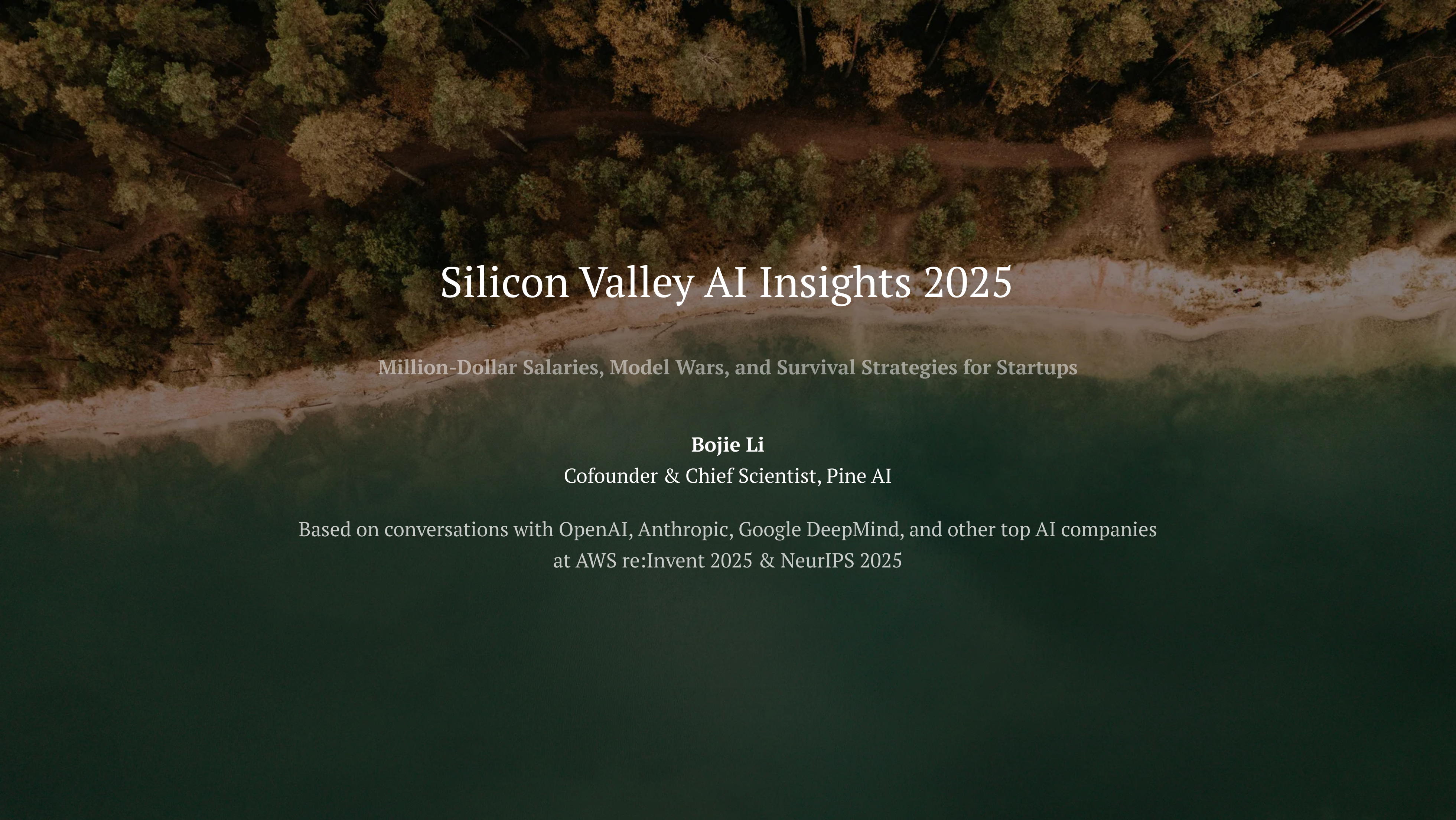


Silicon Valley AI Insights 2025

Million-Dollar Salaries, Model Wars, and Survival Strategies for Startups

Bojie Li
Cofounder & Chief Scientist, Pine AI

Based on conversations with OpenAI, Anthropic, Google DeepMind, and other top AI companies
at AWS re:Invent 2025 & NeurIPS 2025

What We’ll Cover

Technical Insights

- **Vibe Coding**: Where AI shines and where it doesn’t
- **Context Engineering**: Managing context rot
- **File Systems**: As agent interaction buses
- **Scaling Law**: Frontline vs. top scientists’ views

Industry & Strategy

- **Silicon Valley Giants**: Google, OpenAI, xAI, Anthropic
- **Compensation Wars**: Million-dollar salaries
- **Application Development**: Scientific methodology
- **Startup Strategies**: Survival in giants’ shadows

Part I: Vibe Coding

💻

Vibe Coding: Two Extremes

AI coding assistance shows polarized effectiveness

🚀 High Efficiency (3-5x)

- **Startup MVP development**
- **One-off scripts**
- **CRUD/boilerplate code**
- **Data processing**

Clear requirements, standard tech stacks

🐌 Limited Effect

- **Research code**
- **Core infrastructure**
- **Complex refactoring**
- **Big tech daily work**

Deep understanding needed, high coordination costs

Scenario 1: Startup MVP (3-5x Efficiency)

Why it works:

🎯 Perfect Conditions

- **0 to 1 prototype development**
  - Speed matters more than perfection
  - Finding PMF is the priority
  - Can ship daily/weekly for fast iteration
- **Simple tech stack**
  - React, Django, FastAPI
  - Abundant training data
  - Lots of boilerplate

👥 Small Team Benefits

- Low communication overhead
- Fast decisions
- Simple code review
- No cross-department coordination

📝 Typical Tasks

- CRUD business logic
- Simple API development
- Frontend forms/pages
- Data processing scripts

Scenario 2: One-off Scripts (Universal 3-5x)

Works for everyone, including OpenAI & Google researchers

🔧 One-off Scripts

- Data analysis scripts
- Data migration scripts
- Batch processing tools
- Use-and-discard code
- Low quality requirements

🔗 Glue Code (Boilerplate)

- Configuration files
- Data transformation layers
- API call wrappers
- Test case generation

Why it works:

- Clear task boundaries
- No deep business logic understanding needed
- Limited impact if errors occur
- **Even top AI researchers use AI extensively for these**

Scenario 3: Big Tech Daily Work (Limited Gain)

Why efficiency gains are modest:

⏰ Time Distribution

- 30% Meetings
- 20% Coordination
- 20% Documentation
- 15% Debugging
- **15% Coding**

AI only optimizes the last 15%

🏗️ System Complexity

- Deep architecture understanding
- Multi-team codebases
- Backward compatibility
- AI easily introduces regressions

📋 Strict Reviews

- Multiple rounds of code review
- Various lints and tests
- Long deployment process
- Even if AI writes fast, process time remains

Good for: Refactoring repetitive work, test case supplements, simple bug fixes, documentation

Scenario 4: Research Code (Almost Useless)

Why AI can’t help much:

🧠 Intelligence-Intensive

```
# Modifying attention mechanism
# Adjusting training algorithms
# Data ratio optimization

# Might only change 3 lines,
# but requires hours of thinking
```

- Frontier research, not in training data
- Requires deep theoretical background
- Needs innovative thinking
- **AI doesn’t understand it either**

🎨 Highly Customized

- Every research project is unique
- No similar examples to reference
- **Thinking time >> Coding time**
- The bottleneck is **insight**, not typing

💡 **Key Insight:**
When the hard part is figuring out **what** to write, not **how** to write it, AI can’t help.

Scenario 5: Core Infrastructure (Can Hurt)

Why to avoid AI for critical systems:

⚠️ High Risk Areas

- Database schema changes
- Authentication/authorization systems
- Core API contracts
- Payment processing
- Distributed systems coordination

🚨 Problems

- Subtle bugs with serious consequences
- Hard to test comprehensively
- AI lacks system-wide context
- Security vulnerabilities
- Data corruption risks

✅ Better Approach

1. Design by senior engineers
2. Careful manual implementation
3. Extensive review
4. AI can help with: Documentation, Test cases, Boilerplate, etc.

Best Practices from Silicon Valley

What top AI teams actually do:

📏 PR Size Limits

- **< 500 lines per PR**
- Easier to review
- Easier to revert
- AI-generated code is easier to review in small chunks

🤖 Multi-Agent Workflows

- Fully automated agent collaboration
- Code generation → Review → Test → Fix
- Minimal human intervention

🧪 Test-Driven Quality

- Generate tests first
- AI writes code to pass tests
- Automated regression detection
- Continuous validation

🏗️ Large Refactoring

- Break into small, independent PRs
- Each PR is self-contained
- Merge frequently
- Avoid long-lived branches

Code Ownership Principle

"If AI wrote it, but you shipped it,
it’s YOUR code."

🎯 What This Means

- You must understand every line
- You’re responsible for bugs and issues
- Don’t blindly trust AI output
- Review AI code as carefully as junior developer code
- **AI is a tool, not a scapegoat**

Part II: Scientific Application Development

🔬

Top AI Companies’ Methodology

Key insight: Treat AI development like science, not just engineering

📊 Evaluation System

Rubric-based evaluation for every feature

🔄 Data Flywheel

Automated iteration and continuous improvement

👥 Team Separation

Foundation model vs. application teams

Rubric-Based Evaluation

Core principle: Measure everything objectively

📋 What is a Rubric?

A detailed scoring framework that breaks down quality into measurable criteria

```
Example: Code Generation Quality Rubric

1. Correctness (0-40 points)
   - Compiles without errors (10)
   - Passes all test cases (20)
   - Handles edge cases (10)

2. Code Quality (0-30 points)
   - Follows style guide (10)
   - Proper error handling (10)
   - Good variable names (10)

3. Efficiency (0-30 points)
   - Time complexity (15)
   - Space complexity (15)
```

Evaluation System Components

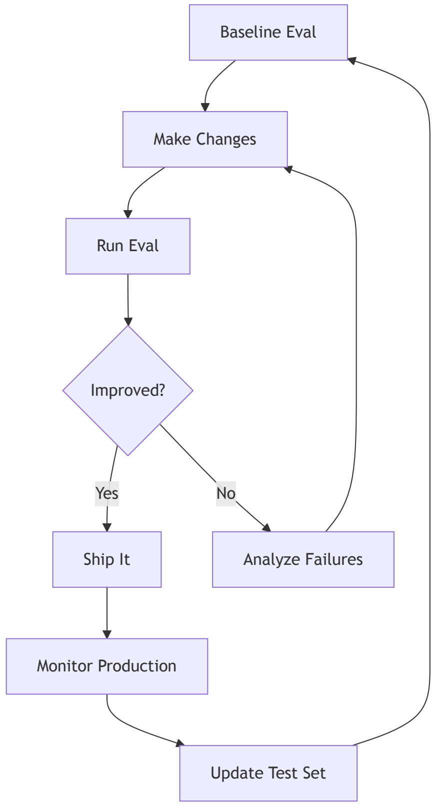
🎯 Test Dataset Construction

- **Real user queries** (anonymized)
- **Edge cases** identified from failures
- **Adversarial examples** to prevent over-fitting
- **Continuous updates** as product evolves

🤖 Automated Grading

- LLM-as-judge with rubrics
- Deterministic checks (regex, exact match)
- Unit test pass rates
- Human eval for ambiguous cases

📈 Continuous Iteration



Foundation Model vs. Application Teams

Model team priorities & app team limitations

🧠 Model Team Priorities

Top priorities:

1. **Four major benchmarks:**
   - Math
   - Coding
   - Computer Use
   - Deep Research
2. **Long-term model capability:**
   - Intelligence improvement
   - General capability enhancement
3. **Vertical domain needs:**
   - Very low priority
   - Basically not responded to

🎨 App Teams ≈ External Startups

Common limitations:

- Use same base model API
- Cannot influence model training direction
- Cannot request targeted optimization

App team’s 2 advantages:

- Can more easily get model team to review prompts, improve context engineering
- Token costs use internal pricing, much cheaper than external API calls (e.g., Cursor API cost >> Claude Code subscription)

External startup advantages:

- Can use multiple models (Google internal only Gemini)
- Can do Mashup Agents: OpenAI + Anthropic + Gemini
- More flexible tech choices

Part III: Silicon Valley Giants

🏢

Google DeepMind: Strengths (1/2)

✅ 1. Leadership Commitment

- **Sergey Brin returned** after ChatGPT launch
- **Demis Hassabis** (DeepMind CEO): strong in both tech and management
- Can unite thousands of smart people
- Avoids serious internal conflicts and politics
- Contrast: Meta (Zuckerberg delegates AI), Microsoft/Apple (execs have limited AI understanding)

Why Gemini App merged into DeepMind?

- Building apps is essentially Research
- Needs scientific methodology, extensive experiments, data-driven approach
- Needs comprehensive Evaluation system
- Aligns with Research thinking mode

✅ 2. Compute Dominance

Hardware advantage:

- TPU + GPU dual track
- Self-developed TPU with continuous capacity
- Years of Nvidia GPU procurement
- Total compute possibly several times OpenAI's

Model scale advantage:

- OpenAI main models: GPT-4o series, hundreds of B params
- Google: Gemini 2.5/3 Pro: Trillions of params (10x larger)
- Gemini 3 Flash params ≈ GPT-4o

Why OpenAI doesn't use bigger models?

- Not enough compute for training & serving
- Too many users (ChatGPT > 1B users)
- Gemini ≈ 600M users, API calls ≈ 1/5 of OpenAI

Google DeepMind: Strengths (2/2)

✅ 3. Human Resources

Massive team scale:

- Nano Banana Pro (Gemini image gen):
  - Algo team: < 10 people
  - Data + Infra team: ≈ 1000 people
- OpenAI equivalent:
  - Algo: < 10 people
  - Data + Infra: **order of magnitude less**

Key advantage:

- Can construct massive domain-specific training data
- Example: Schematics, 9-grid images, etc.
- Requires manual labeling and data construction
- Today's base models still highly dependent on human-constructed data

✅ 4. Ecosystem Moat

- **Chrome browser:**
  - Gemini button integrated in top-right
  - Better than ChatGPT Atlas, Perplexity Comet
  - Can directly ask about current page, summarize long articles
- **Workspace integration:**
  - Google Calendar: Gemini can schedule
  - Google Drive: Gemini can read/write docs
  - Gmail: Gemini can handle emails
  - Natural user base
- **YouTube data:**
  - Years of video data accumulation, multimodal training resource
- **Search engine:**
  - Google Search shows AI Summary

Google DeepMind: 2 Weaknesses

🐌 Big Company Inefficiency

- Multiple approval layers
- Long launch cycles
- Risk-averse culture
- Internal politics
- Coordination overhead

Example: A feature that takes startup 2 weeks might take Google 6 months

🎯 Only General Use Cases

- Won't do vertical/niche optimizations
- Focus on broad applicability
- "One size fits all" mentality
- Doesn't optimize for specific industries

Opportunity: Leaves room for vertical-specific startups

OpenAI: Anxiety & Resource Constraints

😰 Sources of Anxiety

1. Competition Intensifying

- Google, Anthropic, xAI catching up
- Open-source models improving
- Lead is narrowing

2. Talent Structure Issues

- Too many research scientists
- Not enough engineers
- "Academic culture" slowing down
- Focus on papers vs. products

💰 Resource Constraints

The Dilemma:

- Most users: > 1B users
- Limited compute: Not as rich as Google
- Must balance: Training vs. Serving

User experience compromises:

- Early ChatGPT Plus ($20/mo): Violently truncated context
- Context window only 32k tokens
- Severe hallucinations: Lost previous context, later all nonsense
- Author’s experience: Upload book for summary, first few pages OK, rest hallucination

Model routing controversy:

- GPT-5 auto-router: small problems → small model
- Routing inaccurate: important problems → small model
- Users can’t see which model is answering
- Experience degraded, many complaints

xAI (Elon Musk): Extreme Execution

"No Research Engineers, only Engineers"

⚡ Philosophy

- **Results > Research**
- Ship fast, iterate fast
- No academic baggage
- Extreme accountability
- Zero tolerance for underperformance

💪 Work Intensity

- **70+ hours/week** for everyone
- Weekend work expected
- "All-in" mentality
- Not for everyone

🎯 What This Means

Good:

- Extremely fast execution
- No bureaucracy
- High iteration speed
- Strong results-driven culture

Bad:

- Burnout risk
- High turnover
- Less fundamental research
- Work-life balance? What’s that?

Anthropic: Focus on Coding & Agents

🎯 Strategic Focus

- **Constitutional AI** (core differentiator)
- **Code generation** (challenging OpenAI)
- **Agentic systems** (future bet)
- Claude Code, Claude Computer Use
- Context Engineering expertise

🧠 Technical Philosophy

- Safety-first approach
- Long context windows (200K)
- Interpretability research
- Scientific methodology

💡 Key Innovations

1. Context Engineering

- Best practices for long contexts
- Sub-agent architectures
- Skills system
- Progressive disclosure

2. Agentic Tools

- Claude Code
- Agent SDK
- MCP protocol

3. Developer Focus

- Excellent documentation
- Strong developer relations
- Clear best practices

The Talent War: Million-Dollar Salaries

AI compensation has reached unprecedented levels

💰 Salary Tiers (Annual USD)

Fresh Top PhD:

- Total comp: **$1.5M - $2M**
- Conditions: High research level
- Mostly Options (but OpenAI is big enough to cash out)

Experienced AI Engineers:

- Total comp: **$4M - $5M**
- Conditions: Experience at top AI companies or notable academic work

Meta Super Intelligence Level:

- Total comp: **$10M+/year**
- Mostly Meta stock (can be cashed)

📊 AI vs Non-AI Gap

AI Engineers $1M+/year

Non-AI Engineers $250K - $300K/year (Google normal)

3-4x gap for same level

First wave: Meta's Super Intelligence team started crazy hiring, lifted entire market salary levels

Beyond Salaries: Marketing Wars

Massive spending on visibility

💸 Billboard Wars

San Francisco Airport highway:

- Tens of kilometers of billboards
- Hundreds of AI company ads
- Traditional companies (Snowflake) also claiming AI

Funny example - Redis:

> "My boss really wants you to know we're an AI company"

Everyone rushing to align with AI

🔥 AI War vs. Group-Buying War

Group-Buying War (Internet Era):

- Money spent on: Operations, sales, ads
- Core: Grab market, merchants, users
- Human wave tactics

AI War:

- Money spent on: Top talent + GPUs
- Core: Train models, compute power, talent
- Elite tactics, best research team wins

Resource scale:

- 500-1000 GPUs per person
- $1M+ salary per person

Resource Constraints at Foundation Model Companies

Why they can only work on "big problems"

💰 Cost Structure

Per model training run:

- $10M - $100M+ in compute
- Months of preparation
- Hundreds of GPUs/TPUs
- Massive datasets
- Large teams

Result: Can’t do small experiments

🎯 Must Prioritize

Will do:

- General capability improvements
- Things that serve millions
- Benchmark improvements
- Platform features

Won’t do:

- Niche use cases
- Vertical-specific optimization
- Small market features
- Custom training for < 100K users

Part IV: Scaling Law Perspectives

📈

The Great Divide: Researchers vs. Scientists

Frontline engineers see different reality than top scientists

🔬 Frontline Researchers

"Scaling Law is NOT dead"

- Working on daily model improvements
- See continuous gains from scaling
- Optimistic about further progress
- Engineering solutions working well

From: OpenAI, Anthropic, Google DeepMind employees

🧠 Top Scientists

"We need new paradigms"

- Ilya Sutskever
- Richard Sutton
- Yann LeCun

Concerns:

- RL Sampling Efficiency
- Model Generalization
- Continual Learning (ideal form still in research)

Why the Divergence?

Frontline researchers: Top scientists are relatively detached from engineering practice

🎓 Problems Scientists Point Out

Important issues:

- RL Sampling Efficiency problem
- Model Generalization problem

These are real and important

🔧 Engineering Solutions

1. Sampling Efficiency → Use compute

- RL sampling efficiency much worse than supervised learning
- But with enough compute, brute force sampling works
- This is why top companies buy tons of GPUs

🛠️ Generalization Solutions

Engineering methods:

- **Midtrain / SFT:** Manually construct high-quality domain data for continued training
- **Domain datasets:** Collect and label data for specific scenarios
- **Sim Env:** Build simulation environments for controlled learning
- **Rubrics-based Reward:** Reward based on scoring rubrics, not simple binary feedback

Not a silver bullet, but works in practice for many real-world domains

Frontline consensus: Both Pretrain and Posttrain haven't hit ceiling. Release every 6 months with clear capability jumps. Still lots of room for

Consensus: Test on Small Models First

Universal practice across all companies

🔬 Standard Workflow

```
1. Idea → 2. Small model test → 3. Verify improvement → 4. Scale up
```

- Small models (1B-7B parameters) are cheap to iterate
- Can run 100s of experiments for cost of 1 large run
- Most improvements transfer to larger models
- Failures fail fast and cheap

✅ If it works at small scale:

- Very likely to scale up
- Improvement may even amplify
- Worth investing in large run

❌ If it doesn't work at small scale:

- Almost never works at large scale
- Save millions in compute
- Back to drawing board

Part V: Context Engineering

🔧

What is Context Engineering?

"The discipline of optimizing the utility of tokens against the inherent constraints of LLMs"

System Prompt

"Say less, mean more"

Minimal, precise instructions

Tools

"Every tool earns its place"

Self-contained, clear purpose

Data Retrieval

"Load what you need, when you need it"

JIT Context

Long Horizon

Compaction, note-taking, sub-agents

Data Retrieval: The Paradigm Shift

Old: Pre-Loading (Traditional RAG) → New: Just-In-Time

Lightweight Identifiers

- Pass IDs, not full objects
- Agent requests details if/when needed
- **Example:** `user_id: "12345"` → (agent calls `get_user()` if needed) → Full profile

Progressive Disclosure

- Start with summaries
- Agent drills down as needed
- **Example:** File list → File metadata → File contents

Autonomous Exploration

- Agentic Search: Give discovery tools, not data dumps
- Agent navigates information space
- **Example:** `search_docs()` + `read_doc(detail_level)` vs loading all docs

💡 **"Don’t send the entire library. Send a librarian."**

Context Window & Context Rot

Context Window

All frontier models have a maximum number of total tokens able to be processed in a single exchange

Anthropic's context window: 200k tokens

Context Rot

As context grows, output quality regresses

Four types:

1. 🧪 **Poisoning:** Conflicting info corrupts reasoning
2. 📄 **Distraction:** Irrelevant info diverts attention
3. ❓ **Confusion:** Similar items become ambiguous
4. ⚠️ **Clash:** Instructions contradict each other

Research: All models see performance degradation over long contexts
(Chroma Technical Report: Context-Rot)

Three Strategies for Long-Horizon Tasks

Compaction

- Periodically summarize intermediate steps and/or compress history
- Reset context with compressed summary
- Retain only essential information
- **Trade:** Minor detail loss for continued operation
- **Example:** *"User wants X, tried Y, learned Z"* vs. full conversation

Structured Memory/Note-taking

- Agents maintain explicit memory artifacts (external persistent storage)
- Store "working notes": decisions, learnings, state in structured format
- Retrieved on-demand rather than kept in context
- **Example:** Decision log, key findings document

Sub-Agent Architectures

- Decompose complex tasks into specialized agents
- Each sub-agent has focused, clean, narrow context
- Main agent orchestrates and synthesizes results
- **Example:** Code-review agent spawns doc-checker sub-agent

Skills: Progressive Disclosure in Action

Skills are organized folders of instructions, scripts, and resources that Claude can discover and load dynamically

pdf/SKILL.md

```
YAML Frontmatter

name: pdf
description: Comprehensive PDF toolkit for extracting text anc
merging/splitting documents, and filling out forms.
```

```
## Overview

This guide covers essential PDF processing operations using Py
and command-line tools. For advanced features, see `/reference
If you need to fill out a PDF form, read `/form.md` and follov
```

pdf/reference.md

# PDF Processing Advanced Reference

This document contains advanced PDF processing features...

pdf/forms.md

If you need to fill out a PDF form, first check to see...

Discovery: Claude navigates and discovers added detail as needed
Executable Scripts: Token efficient for operations better accomplished by traditional code

Tool Design Best Practices

Elements of Strong Tool Design

- Use a **simple & accurate tool name**
- **Detailed and well-formed descriptions**
  Include what the tool returns, how it should be used, etc.
- **Avoid overly similar tool names or descriptions!**
- **Tools that perform one action work better**
  Try to have at most 1 level of nested parameters
- Provide examples - expected input/output format
- **Pay attention to the tool results’ format**
- **Test your tools!** Make sure agents use them well

```json
{
  "name": "search_customers",
  "description": "Search customer database by
  name, email, or ID. Returns matching customer
  records.",
  "input_schema": {
    "type": "object",
    "properties": {
      "query": {
        "type": "string",
        "description": "Search term (name,
  email, or customer ID)"
      },
      "max_results": {
        "type": "integer",
        "default": 10,
        "description": "Number of results to
  return (default: 10, max: 50)"
      }
    },
    "required": ["query"]
  }
}
```

Effective Context Engineering Benefits

Handle context window limits

→

Reliability

Reduce context rot

→

Accuracy

Optimize for prompt caching

→

Cost & latency

File System as Agent Interaction Bus

Anthropic’s core view: Coding Agent is the foundation of all general-purpose Agents

❌ Tool Call Problems

Outputting large content unstable:

- Tool call outputs hundreds of lines
- If interrupted midway, all work lost
- Cannot recover

Not iterable:

- Output is output, cannot modify
- Want to change one section? Regenerate all
- Can’t do "draft-revise-finalize" flow

✅ Advantages of File System Abstraction

Broad "Coding":

- Includes docs, reports, any structured content

Persistent:

- Written to file, content saved
- Even if agent crashes, file remains

Iterable:

- Read file, modify part
- Multiple revisions, gradually improve
- Just like humans writing docs

Universal:

- `ls` , `read_file` , `write_file` , `edit_file` , `delete_file`
- All SOTA LLMs understand these operations

Part VI: Startup Strategies

🚀

What Startups Should AVOID

❌ Don’t Fight Head-On with Base Model Companies

Startups should NOT touch: General Coding Agent, Deep Research, Computer Use

Why startups will lose:

- **Can’t hire:** People who truly understand model training are extremely expensive. Startup raises millions to tens of millions USD, hire a few people, money gone.
- **Not enough compute:** Training a general model, even post-training, needs hundreds to thousands of GPUs. Startups can’t rent that many.
- **Not enough data:** General capabilities need massive high-quality data. Big tech has ecosystem advantages (YouTube, Web Search). Startups can’t get this data.

❌ Don’t Casually Touch Model Training

1. **People who truly understand models are too expensive**
   - They’re all at big tech, won’t easily leave
   - Startups simply can’t poach them
   - Model training needs lots of trial and error – significant cost
2. **Open-source + finetuning dilemma**
   - Open-source model quality 2 orders of magnitude worse than closed-source
   - Finetuning hard to bridge this gap, unless extremely niche vertical domain

When to consider training:

- Only for specific scenario small models (8B/32B)
- Domain is quite niche, general models insufficient

What Startups SHOULD Do

✅ Go Vertical + Master Context Engineering

Find your niche and build deep expertise:

- Domain expertise + Customer access
- Specialized data + Understanding of workflows

Context Engineering is HARD - requires serious expertise:

- Expert-level domain-specific prompts
- Specialized tools and sub-agent architectures
- Domain-specific evaluations and iterations

✅ Build Domain Knowledge Base

Critical foundation:

- Curate high-quality domain-specific data
- Build proprietary knowledge graphs
- Integrate industry best practices

🔄 Feedback loop for real business scenarios

- Capture user interactions and corrections → Learn from production failures
- Improve prompts based on real data → Refine knowledge base continuously
- **The flywheel:** More users → More feedback → Better context engineering → Better performance → More users

Startup Talent Strategy

Core principle: Hire smart, strong learners WITHOUT AI background

🎯 Why This Strategy Works

1. AI evolves too fast, weak compounding:

- Best practices change every 3-6 months
- Past experience depreciates quickly

2. New people reach frontier fast:

- Smart + strong learning ability + willing to dig deep
- 6-12 months → above-average industry level
- No PhD needed, no big-tech background needed

3. Massive cost advantage:

- **5-10x cost difference**

💰 What You Can’t Match

You can’t compete on:

- $1M - $10M salaries
- Prestigious brands (OpenAI, DeepMind)
- Massive compute budgets
- Large teams

What You CAN Offer:

- Ownership & significant equity
- Real impact on product
- Learning & growth
- Close to customers
- Mission & culture
- Fast iteration, no bureaucracy

Wait for Your Wave

🌊 Key: Stay Relevant

1. When there’s no wave, don’t give up:

- Solidly build your foundation
- Don’t think CV/NLP expertise is useless in LLM era
- Don’t give up because you don’t see opportunities

2. Prepare your team:

- Strong engineering capability
- Strong learning ability
- Strong cohesion

3. Stay Relevant:

- First Principle Thinking
- Closely track frontier products and research

⏰ Stay Ahead of the Curve

4. Predict trends:

- Model and product trends
- When wave comes, you’re ready

5. Catch your wave:

- Can you quickly build and go to market?
- When users grow, is your data flywheel ready?
- ChatGPT, Gemini App, Cursor all have data flywheels

When your wave comes, are you prepared?

Part VII: Key Q&A Insights

💡

Q1: Finetune or Use Closed-Source Models?

Conclusion: Closed-source + Context Engineering > Open-source + Finetune

📊 Three Key Gaps

1. Knowledge Density Gap:

- Closed-source models lead by 2 generations
- Higher quality training data
- Better parameter efficiency

2. Reasoning Density Gap:

- Open-source (Qwen, Kimi) needs very long CoT
- Relies on longer thinking time for results
- Closed-source CoT is more compact and efficient

3. Generalization Gap:

- Open-source optimized for public benchmarks
- Weak generalization on non-benchmark scenarios

✅ When to Use Open-source + Finetune

1. Extremely niche domain:

- Almost no data on public internet
- Must inject domain knowledge via finetuning

2. Data privacy requirements:

- Cannot send data abroad
- Must deploy locally

3. Extremely cost-sensitive:

- Huge volume, API costs unsustainable
- Self-hosting open-source is more economical

Q2: Is Personalization a Real Problem?

Yes, and it’s a core competitive advantage for the future

🎯 Why It Matters

Recommendation Systems Evolution:

- Traditional: Everyone reads same newspaper
- ByteDance: Everyone sees different content
- **Insight:** People live in different worlds with different values
- Personalized products are more human-centric

AI’s Future:

- Not just one "Universal Value"
- Should adapt to each user’s values and preferences
- Detailed value differences matter

🔧 Technical Challenges

1. Factual Information (Easy)

- Birthday, address, card numbers, work info
- Just remember, no ambiguity
- We’re already good at this

2. User Preferences (Very Hard)

- **Context-dependent:**
  - Academic format for papers ≠ travel guides
  - AI easily over-generalizes preferences
- **One-time vs. Long-term:**
  - "I ordered Sichuan food yesterday"
  - ≠ "User loves Sichuan food"
  - Maybe friend’s preference, or one-time thing

Q3: Edge-Cloud Agent Coordination?

Future: Inevitable, but core challenge is APP state sync

⚠️ Core Challenge: APP State Sync

Login state problems:

- User logged into WeChat on phone
- Cloud agent also needs to login?
- WeChat only allows one client → kick each other off

Fraud detection issues:

- Cloud IP ≠ Phone IP, may be in different countries
- Triggers fraud detection → account banned

Repeated login hassles:

- Need to re-login to all apps in cloud
- Poor user experience
- Privacy concerns

✅ Solution: Mirror System

Douyin Phone's approach:

- "Shadow system" on local device
- Agent operates apps in background
- Apps think they're in foreground
- User operates real foreground
- Two systems parallel, non-interfering

Ideal solution (needs OS support):

- Android/iOS/HarmonyOS system-level
- Sync APP state to cloud
- Cloud agent operates cloud APP mirror
- Sync back to phone when needed

Two Independent Dimensions: Agent location (edge/cloud) × Model location (edge/cloud)

Q4: Best Practices for Using AI to Write Code?

Core principle: Humans must understand more than AI

🎯 Why?

- You must review AI's code
- AI makes mistakes, humans find them
- AI can find simple syntax errors
- But complex architecture issues? AI struggles

👔 Role Transformation

From: Coder
To: Architect + Reviewer

🔄 New Workflow

1. **Requirements breakdown (Human)**
   - Break into small tasks (< 500 lines)
2. **Coding (AI)**
   - Can parallelize multiple AIs
3. **Auto Testing (AI)**
   - Run test suites
4. **Code Review (AI)**
   - 5-6 different Review Agents
   - All pass → generate PR
5. **Final Review (Human)**
   - Approve or request changes

Q5: How to Ensure Workflow Stability When Model Updates?

🧪 Why Evaluation Matters

1. Models upgrade:

- Base models update every few months
- Each upgrade may break existing workflows
- Need rapid compatibility validation

2. Prompts adjust:

- Frequently need to optimize prompts
- Change A, might break B
- Need comprehensive regression testing

3. Avoid subjective judgment:

- Can’t rely on manual testing a few examples
- Time-consuming and not objective
- Easy to miss problems

✅ Complete Evaluation System

1. Test Dataset:

- Extract representative cases from production

2. Rubric-based Assessment:

- Not just overall good/bad
- Break into multiple sub-metrics
- Score each independently

3. Automated Execution:

- Switch model, run tests
- Human review report

4. Continuous Optimization:

- Find new issues, add to dataset
- Form closed loop

Q6: How to Get AI Frontier Information?

Most recommended: X (Twitter)

🐦 Why X (Twitter)?

- China's "Top 3 AI Media" (新智元, 机器之心, 量子位) all get news from X
- First-hand papers and technical discussions
- Many top researchers share there

How to Use X?

- Follow technical leaders
- Follow accounts that specifically share papers

📚 Other Sources

Academic:

- arXiv daily digest
- Top conferences (NeurIPS, ICML, ICLR)

Industry:

- Company blogs (OpenAI, Anthropic, etc.)
- Follow researchers on social media

Community:

- AI Discord servers
- Local meetups
- Conferences

Q7: Multi-Goal Prompt Conflicts?

"When I ask for A and B, I only get A"

❌ Common Problem

```
Write code that is:
- Fast
- Readable
- Well-documented
- Secure
- Minimal dependencies
```

Model focuses on first 1-2, ignores rest

✅ Two Solutions

1. Evaluation to Prevent Regression

- Change prompt → comprehensive testing
- Ensure no breaking of other features
- Data-driven, objective assessment

2. Structured Prompt Organization

❌ Don’t pile up rules:

- 101 rules, keep adding

✅ Write like a book:

- Hierarchical structure
- Like new employee handbook
- Logical guidance with considerations

Key Takeaways

🎯

Summary: What We Learned

💻 Technical

1. **Vibe Coding works for:**
   - Startups, scripts, boilerplate
   - **Not for:** Research, core infra
2. **Context Engineering is critical:**
   - Dynamic prompts
   - Sub-agents
   - File system as bus
3. **Scaling Law debate:**
   - Engineers: still working
   - Scientists: need new paradigms

🏢 Strategic

1. **Giants have advantages:**
   - Compute, talent, ecosystem
   - But slow and generic-focused
2. **Startups should:**
   - Go vertical
   - Master context engineering
   - Avoid benchmark competition
3. **Development is science:**
   - Evaluation systems
   - Data flywheels
   - Continuous iteration

Final Thoughts

The AI Revolution is Still Early

🌊 Find Your Wave

Don’t compete where giants dominate. Find niches they can’t reach.

🔧 Master the Tools

Context engineering and evaluation systems are your moat.

🚀 Move Fast

The field changes weekly. Iterate quickly, learn constantly.

Thank You

Bojie Li
Cofounder & Chief Scientist, Pine AI

Based on conversations with OpenAI, Anthropic, Google DeepMind, and other top AI companies
at AWS re:Invent 2025 & NeurIPS 2025

Questions?