Beyond Prompts: How to Build Real AI Products

Agenda

GenAI Fundamentals

Market landscape and valuations

LLMs, tokens, pricing

Training stack & capabilities

Products & Ops

Agents, tools, orchestration

Low-code builders, MCP

Toolkits & operating practices

Inside AIPM

Context engineering, RAG

Prompt strategies, evaluation

Fine-tuning & decision flows

Also in this session

Case study: Granola

Example: Notebook LM

Key insights • Glossary • Q&A

Section I

GenAI Fundamentals

What’s driving adoption, how LLMs work, training stack, and core capabilities

GenAI Market: Scale & Momentum

Valuations and growth since 2022 (illustrative figures from session)

OpenAI

$500B

Valuation

Anthropic

$183B

Valuation

xAI

$200B

Valuation

Microsoft

$1.8T → $3.8T

Valuation growth

Google

2.5×

Growth

Meta

$300B → $1.8T

Valuation growth

Startups: Perplexity $15B | Glean $7B | 11 Labs $6B | Nvidia ~10× growth

Why now: proven daily utility vs. prior hype cycles (e.g., blockchain/NFT).

Next-Token Prediction

- Given prior tokens (context), the model predicts the most likely next token, repeatedly forming text.
- Tokens are numeric units of text; roughly 2 words ≈ 3 tokens.
- Outputs are probabilistic (sampling from a distribution), so the same prompt can yield different answers.

GPT-4 vocab ~100k tokens

Pricing ≈ $1.25 / 1M tokens

Tokens & Pricing Essentials

Key metrics and practical cost-control tips

Unit
Tokens (input + output)

Rule of thumb
~2 words ≈ 3 tokens

Vocabulary size
GPT-4 ~100k tokens

Typical pricing (model-dependent)

≈ $1.25 per 1M tokens

Billable = input + output

Track usage by endpoint

Practical tips

- Optimize prompts (be concise; specify format)
- Use RAG to shrink context to only relevant chunks
- Cache & reuse results; avoid repeated calls

Measure token usage per feature to guide optimization.

Training Stack: Pre-training → Fine-tuning → RLHF

Three stages to build helpful, safe, and capable LLMs

Pre-training

Unlabeled internet-scale text • Next-token objective

Learns general language patterns and knowledge

Fine-tuning

Labeled input-output pairs • Task/domain adaptation

Improves relevance, tone, formatting for use cases

Post-training (RLHF)

Human feedback • Safer, more helpful behavior

Reduces hallucinations, aligns with user intent

Transformers (2017 “Attention Is All You Need”) enable parallel attention over tokens.

GPUs provide the parallel compute required for training and inference at scale.

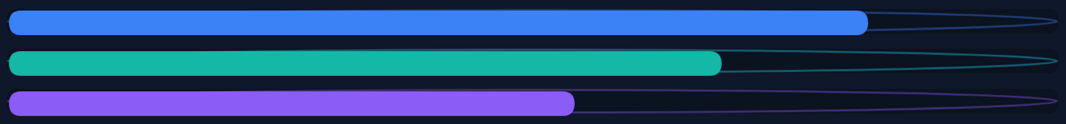
Compute & Output Nature

GPUs enable parallelism • LLM outputs are probabilistic

Compute: GPUs & Parallelism

- GPUs run massive parallel math, accelerating attention operations in transformers.
- Enables feasible training/inference at scale with large token contexts.
- Throughput and latency depend on model size, batching, and hardware.

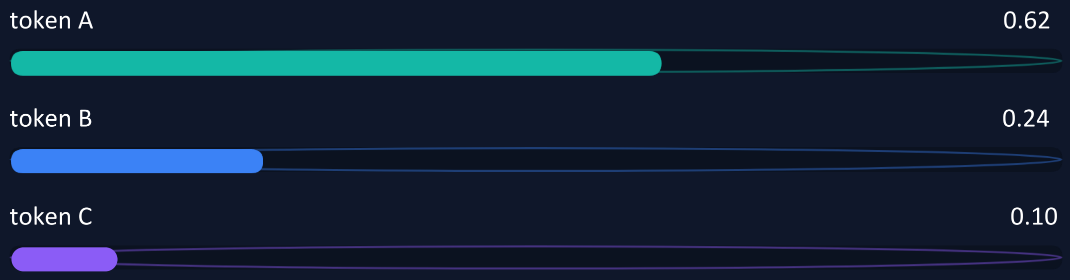
Parallel lanes (illustrative)



Outputs: Probabilistic Sampling

- LLMs return a distribution over the next token; responses may vary for the same prompt.
- Control behavior using temperature, top-p, and system prompts.
- For reproducibility, fix seeds where supported and constrain output formats.

Next-token probabilities (illustrative)



Temperature | Top-p | System prompt

What LLMs Can Do

Three core capabilities that power intelligent experiences

Understand

- Interpret intent and context
- Classify, extract entities, detect sentiment
- Ground responses with provided context

Transform

- Summarize, rewrite, translate
- Answer questions over documents (RAG)
- Generate or refactor code with constraints

Generate

- Create net-new text, outlines, and drafts
- Produce images or audio via connected tools
- Synthesize plans, ideas, and code snippets

Shift: from CRUD systems to intelligent workflows with a reasoning “brain”. Combine capabilities for end-to-end experiences.

Section II

Inside AIPM

From context engineering and RAG to prompts, fine-tuning, and decision frameworks

Case Study: Granola (AI Meeting Assistant)

User value • Business value • Design value • Key features

User Value

- Saves time with automatic notes and summaries
- Improves accuracy vs. manual note-taking
- Lets participants focus on the discussion

Business Value

- Large productivity market (meetings at scale)
- Strategic fit with Zoom/Google Meet ecosystems
- Recurring value drives retention and upsell

Design Value

- Minimal friction: auto-join, seamless transcript
- Clear summaries with action items
- Privacy controls and easy sharing

Key Features (Engineering)

- Audio capture via meeting APIs or bots
- Speech-to-Text (Whisper/Google)
- LLM system prompts for summaries + action items
- Growth: bot visibility and post-meeting email recaps

Start from user and business value, then design for minimal friction; implement the simplest reliable pipeline first.

Granola Architecture — Audio → STT → LLM Summary

Simple, reliable pipeline powering meeting notes and growth

Audio Capture

- Meeting APIs or bot joins (Zoom, Google Meet)
- Streams audio securely with consent

→

Speech-to-Text (STT)

- Whisper / Google Speech-to-Text
- Returns timestamped transcript

→

LLM Summary + Actions

- System prompts structure notes, decisions, owners
- Produces concise summary + action items

→

Delivery & Storage

- Notes in app; email recap to attendees
- Optional push to calendar/CRM/task tools

Growth Tactics

- Bot presence in meetings → viral loop
- Post-meeting email recaps
- Shareable links & collaboration
- Lightweight onboarding & privacy controls

Note: Obtain consent, provide opt-outs, and secure storage to build trust.

Context Engineering

Principles

- Respect token limits — include only the most relevant context.
- Decompose tasks — structure instructions, steps, and constraints clearly.
- Prefer retrieval (RAG) over dumping large documents into the prompt.

Token budget awareness

Relevance > Volume

Context Window (tokens) — Curate the prompt

System prompt	User query	Retrieved chunks	Headroom

Retrieval-first flow

Docs → Chunk → Embed → Vector DB → Top-k → Build prompt

Retrieve only what’s relevant; cite sources/IDs for traceability.

RAG Pipeline — End-to-End (7 Steps)

Chunk → Embed → Store → Query → Retrieve → Prompt → Generate

Chunk
Split docs into semantic pieces

Embed
Convert chunks to vectors

Store
Vectors in a Vector DB

Query
Embed the user question

Retrieve
Top-k similar chunks

Prompt
Compose: context + question

Generate
LLM answers with citations

Chunking: semantic, ~200–500 tokens

Vector DB: Pinecone, Supabase

Include source IDs for traceability

Prompt Engineering — Core Strategies

Five patterns to increase control, consistency, and quality

Role assignment

Set persona, seniority, and domain expertise.

`Act as a senior product manager`

Output format

Specify JSON, tables, or bullet structure explicitly.

`Return JSON: {"summary": "...", "actions": []}`

Multistep reasoning

Ask to think step-by-step or plan before answering.

`Think step-by-step; list assumptions first`

Few-shot prompting

Provide 1–3 examples to guide style and format.

`Example → Response pairs`

Constraints

Define boundaries: cite sources, avoid speculation, and stick to provided context.

`Do not assume facts not in the context`

Prompt Engineering — Examples & Best Practices

Code-style prompts with operational tips and recommended tools

Example — Structured Summary

```
Act as a senior PM. Summarize the meeting in 5 bullets.
Return JSON with 3 action_items [{ "title", "owner", "due_date" }].

// Constraints: use only transcript; no assumptions
```

Example — Comparison Table

```
Compare RAG vs fine-tuning for knowledge gaps.
Output a 3-row table: criteria | RAG | fine-tuning.
// Include latency, cost, freshness
```

Example — Guardrails

```
Use only provided context. Cite source_ids for facts.
If missing info, reply: "Insufficient context".
```

Best Practices

- Be explicit: role, output format, constraints.
- Induce reasoning: ask to plan or think step-by-step.
- Use few-shot examples to shape style and structure.
- Version prompts and evaluate changes before rollout.

Prompt directory | Versioning | Evaluation

Recommended Tools

Prompt Wildcard
Reusable prompt variables and templates

Anthropic Prompt Creator
Iterate prompts with built-in guidance

Note: Track prompt changes alongside model versions and context configs.

Fine-tuning — Full vs PEFT (LoRA/Adapters)

Adapt base LLMs for domain and style with clear cost/latency tradeoffs

Full Fine-tuning

All params updated

Pros

- Highest task performance ceiling
- Deep domain adaptation and control
- Best for complex, specialized skills

Cons

- Expensive compute and longer training
- More data required; careful safety tuning
- Higher latency/serving cost for large models

Example: BloombergGPT (finance)

Use when: deep domain accuracy is critical

PEFT — LoRA/Adapters

Small subset updated

Pros

- Much cheaper and faster to train
- Lower latency and easier deployment
- Swap adapters per domain/persona

Cons

- May not match full fine-tune peak performance
- Limited for drastic behavior shifts
- Still requires quality labeled data

Example: Bank AI (tone & style)

Use when: efficient domain style customization

Guideline

Start with Prompts → Add RAG for missing knowledge → Fine-tune for domain behavior

Contextualization Decision Tree

When to use Prompt Engineering, RAG, or Fine-tuning

Condition	Recommended Method	Examples
Base LLM sufficient (with clear prompting)	Prompt Engineering	ChatGPT, Granola
Missing or dynamic knowledge	RAG (Retrieval-Augmented Generation)	Notion AI, Notebook LM
Need domain-specific behavior or style	Fine-tuning	Bloomberg GPT

Guideline

Start with Prompts → Add RAG for missing knowledge → Fine-tune for domain behavior

Transfer Learning — Big → Small

Large LLM → synthetic data → small model fine-tuning for cost and latency optimization

1) Large LLM

Frontier model as teacher

- Generate domain-specific synthetic Q/A, code, or labeled samples
- Apply quality filters and instructions

2) Synthetic Data

Curated training corpus

- Format as input → output pairs with rationales when useful
- De-duplication and balance across topics

3) Small Model + Fine-tune

Efficient deployment target

- Fine-tune smaller models (e.g., LLaMA variants, GPT nano-class)
- Use PEFT (LoRA/adapters) for faster, cheaper training

$ Lower cost & latency vs. large models

Competitive quality on narrow tasks

Validate data quality; monitor drift

Use cases: customer support assistants, coding copilots, niche Q&A. Start with prompts → add RAG if needed → fine-tune small models with synthetic data for cost-effective serving.

Section III

Products, Tooling & Ops

Agents, low-code builders, MCP integrations, and evaluation practices

AI Agents — Overview & Components

Definition, agent types, core components, and common platforms

Definition

AI agents connect an LLM to tools/APIs and data so they can take actions autonomously or semi-autonomously to achieve goals.

Workflow

Follows predefined steps or playbooks.

Autonomous

Makes decisions based on context and goals.

Platforms

- Relay.app
- ZPR
- agents.ai
- OpenAI agent kits

Example use case: weekly LinkedIn post summarizer that emails a digest.

Core Components

Reasoning

Plan, decide, and summarize.

Memory

State, long-term notes, vector DBs.

Tools

APIs, web actions, databases.

Orchestration

Multi-step flows, task routing.

Guardrails

Policies, constraints, safe tools.

Tip: Start simple with workflow agents, add memory and guardrails, then iterate toward autonomy as you gather feedback.

Low-Code AI App Builders (Lovable)

Architecture flow and business value — democratizing app creation

Architecture Flow

User Prompt
Describe the app and goals in natural language.

System Boost
Enhance with role, constraints, and style.

LLM Plan
Generate file structure, APIs, and tasks.

Agent Build/Test
Create code, run tests, debug iteratively.

Deploy
App hosted and shareable instantly.

Lovable example: users describe an app, the system enhances instructions, plans the build, agents generate and test code, then deploy a working app — often in minutes.

Business Value

- Democratizes app creation for non-coders, expanding addressable market.
- Rapid prototyping reduces time-to-value for entrepreneurs and teams.
- Freemium + community templates drive viral distribution and retention.

Architecture applies broadly to low-code AI platforms; pair with evaluation and guardrails for production use.

- Faster MVP cycles
- Lower build costs
- Broader creator base

Model Context Protocol (MCP)

Standardized layer for LLM ↔ services: Client • Server • Data Source

MCP Architecture Stack

MCP Client

Hosted on AI platform (e.g., ChatGPT, Claude)

Natural language requests

↓

MCP Server

Hosted by service provider (e.g., Zomato)

Maps intents to capabilities

↓

Data Source / Actions

- Menu & catalog APIs
- Orders & tracking
- Payments

Goal: let LLMs call standardized tools without bespoke API knowledge. Scope capabilities explicitly; log and evaluate calls.

Zomato Example

"Order a veg biryani from Zomato to {Fill in detailed address here}."

Client parses intent → Server maps to "place_order" → APIs create order

Benefits

- Natural language interface
- Faster integrations
- Scoped, safer actions

Implementation tip: convert existing REST endpoints into MCP capabilities with clear schemas and permission checks.

Evaluation — Quality, Safety, Reliability

Assess outputs, reduce risks, and enforce guardrails for production AI

Challenges

- Hallucination — confident but incorrect or unsupported claims.
- Bias — unfair, harmful or non-inclusive outputs.
- Non-determinism — variable answers to the same prompt.

Impact: reduced trust, compliance risk, unpredictable UX.

Methods

- LLM as judge — a stronger model scores relevance, correctness, tone.
- Automated metrics — task-specific checks (factuality, structure, toxicity).
- Manual review — sample audits to calibrate rubrics and catch edge cases.

Golden datasets

A/B comparisons

Score thresholds

Guardrails

- Prompt constraints — roles, style, and “don’t assume beyond context.”
- Tooling limits — allowlisted APIs, safe actions, timeouts.
- Red teaming — adversarial tests; feedback loops to retrain or refine.

Log prompts, context, outputs, and judgments for auditability.

Define task & rubric

Test on datasets

Track scores & regressions

Toolkit + Key Insights

Tools by category • 10 takeaways for AI Product Managers

AI PM Toolkit

Discovery

ChatGPT | Notebook | Perplexity | Mixpanel | TextSQL

Delivery

Jira | Asana | VZero | Postbot | Figma

Distribution

Craftable | Notion | Genex

Key Insights (10)

1. GenAI growth is driven by real utility, not just hype.
2. LLMs are probabilistic; engineer prompts and controls carefully.
3. RAG reduces context size and boosts factual accuracy.
4. Prompting is iterative—maintain a versioned prompt directory.
5. RLHF improves helpfulness, tone, and trustworthiness.
6. Agents extend LLMs with tools to automate real workflows.
7. Low-code builders democratize product creation at speed.
8. MCP standardizes tool access and accelerates integrations.
9. Rigorous evaluation (LLM-as-judge + metrics) is essential.
10. Win by blending tech depth with user empathy and business sense.

Tip: Start lightweight—use prompts and RAG first; fine-tune or build agents only when the use case clearly demands it.

Glossary

Key AI/ML terms used in this masterclass

Term	Definition
LLM	Large Language Model — neural network trained to predict the next token, enabling language understanding and generation.
Token	Numeric unit of text used by models; pricing and context limits are measured in tokens.
Transformer	Architecture (2017) that processes sequences in parallel using attention mechanisms.
Attention	Mechanism that weighs relationships among tokens to capture context.
RAG	Retrieval-Augmented Generation — combines vector search of relevant docs with LLM generation.
Embedding	Vector representation of text that captures semantic meaning for similarity search.
Fine-tuning	Adapting a pre-trained model to a domain/task using labeled examples.
PEFT	Parameter-Efficient Fine-Tuning (e.g., LoRA, Adapters) — updates a small subset of weights.

Closing — Build • Measure • Learn

Action + feedback + reflection → mastery through building

Build. Measure. Learn.

Ship small, gather signal, reflect deeply, then iterate.

Build
Create small MVPs; ship weekly.

Feedback
User signals + LLM-as-judge evals.

Reflection
Retros; refine prompts/RAG.

Mastery
Compound learning via iteration.

Start with Prompts → Add RAG for missing knowledge → Fine-tune for domain behavior

Prioritize user value • evaluate rigorously • iterate fast

Thank you! Questions?