Meeting 13: LLM Agents & Advanced Prompt Engineering

AI-40X: Generative AI & Large Language Models

Hendri Karisma, M.T.
Dosen Teknik Informatika STMIK Tazkia
VP Engineering at jejakin.com, 2026

Semester 4 — 2025/2026

Outline

Apa Itu Prompt Engineering?

- **Prompt** = instruksi yang kita berikan ke LLM untuk mendapatkan output yang diinginkan.
- **Prompt Engineering** = seni dan sains merancang prompt agar LLM menghasilkan jawaban yang akurat, relevan, dan sesuai format.
- Mengapa penting?
  - Model yang sama bisa menghasilkan jawaban sangat berbeda tergantung cara kita bertanya.
  - Prompt yang baik = mengurangi halusinasi, meningkatkan akurasi, menghemat token (biaya).

Zero-Shot vs Few-Shot Prompting

Zero-Shot Prompting

- Tidak memberikan contoh apapun.
- Hanya instruksi langsung.
- Cocok untuk tugas sederhana yang sudah dipahami LLM.

Contoh Zero-Shot

```
Klasifikasikan sentimen kalimat
berikut sebagai positif atau negatif:
"Makanannya enak sekali!"
```

Few-Shot Prompting

- Memberikan beberapa contoh input-output di dalam prompt.
- LLM belajar pola dari contoh tersebut.
- Cocok untuk tugas yang butuh format spesifik.

Contoh Few-Shot

```
"Pelayanannya buruk" -> Negatif
"Harganya murah" -> Positif
"Antriannya panjang" -> ???
```

System Prompt & Parameter Decoding

System Prompt

- Instruksi "tersembunyi" yang mengatur perilaku dan batasan LLM.
- Contoh: `"Kamu adalah asisten akademik STMIK Tazkia. Jawab hanya dalam Bahasa Indonesia. Jangan menjawab pertanyaan di luar topik akademik."`
- Digunakan untuk menjaga *persona*, format output, dan keamanan.

Parameter Decoding

- **Temperature** (0–2): Mengontrol "kreativitas". 0 = deterministik, 1+ = acak.
- **Top-p** (nucleus sampling): Hanya memilih dari token yang probabilitas kumulatifnya $\leq p$.
- **Top-k**: Hanya memilih dari k token dengan probabilitas tertinggi.

Tips

Faktual → temp rendah (0.1–0.3).
Kreatif → temp tinggi (0.7–1.0).

Masalah: LLM Gagal di Penalaran Multi-Langkah

- LLM sangat bagus dalam tugas berbasis pola (terjemahan, ringkasan).
- Tapi sering **gagal** dalam soal yang butuh penalaran bertahap:
  - Soal matematika
  - Logika multi-langkah
  - Perencanaan
- **Penyebab:** LLM menghasilkan jawaban token-per-token; tanpa "ruang berpikir", dia langsung menebak jawaban akhir.

Contoh Kegagalan

Pertanyaan: "Siti punya 23 apel, dia berikan 7 ke Budi, lalu beli lagi 12. Berapa total apelnya?"
Jawaban LLM (tanpa CoT): "35" (SALAH – harusnya 28)

Solusi: Chain-of-Thought Prompting

- **Ide inti:** Minta LLM menunjukkan langkah-langkah penalarannya sebelum menjawab.
- **Zero-Shot CoT:** Cukup tambahkan `"Mari kita pikirkan langkah demi langkah"` di akhir prompt.
- **Few-Shot CoT:** Berikan contoh soal + langkah penyelesaian di prompt.

Zero-Shot CoT

Prompt: "Siti punya 23 apel, dia berikan 7 ke Budi, lalu beli lagi 12. Berapa total apelnya? *Mari kita pikirkan langkah demi langkah.*"

Jawaban LLM:

1. Siti awalnya punya 23 apel.
2. Diberikan 7 ke Budi: $23 - 7 = 16$.
3. Beli lagi 12: $16 + 12 = 28$.

Jawaban: 28 (BENAR)

Self-Consistency: Meningkatkan Akurasi CoT

- **Masalah:** Satu kali CoT bisa saja menghasilkan jalur penalaran yang salah.
- **Solusi (Self-Consistency):**
  1. Jalankan prompt yang sama berkali-kali (misal 5 kali) dengan temperature ¿ 0.
  2. Setiap run menghasilkan jalur penalaran berbeda.
  3. Ambil jawaban yang **paling sering muncul** (majority voting).
- Secara empiris meningkatkan akurasi 5–15% pada benchmark matematika dan logika.

Run	Jalur Penalaran	Jawaban
1	$23 - 7 = 16,\ 16 + 12 = 28$	28
2	$23 + 12 = 35,\ 35 - 7 = 28$	28
3	$23 - 7 = 15$ (salah), $15 + 12 = 27$	27
Majority Vote		**28**

ReAct: Reasoning + Acting

- **Masalah baru:** CoT membuat LLM bisa berpikir, tapi dia masih "terisolasi" – tidak bisa mengakses informasi dunia nyata (internet, database, kalkulator).
- **ReAct (Yao et al., 2022):** Gabungkan *Reasoning* (berpikir) dan *Acting* (bertindak) dalam satu loop.
- Siklus: **Thought** → **Action** → **Observation** → ulangi sampai jawaban ditemukan.

Contoh ReAct Loop

Pertanyaan: "Siapa rektor STMIK Tazkia saat ini?"

Thought 1: Saya perlu mencari informasi terbaru tentang rektor STMIK Tazkia.
Action 1: `search("rektor STMIK Tazkia 2025")`
Observation 1: Hasil pencarian: "Prof. Dr. X menjabat sebagai rektor sejak 2023..."
Thought 2: Saya sudah menemukan jawabannya.
Action 2: `finish("Prof. Dr. X")`

Mengapa ReAct Lebih Baik?

CoT Saja

- LLM berpikir berdasarkan pengetahuan internalnya saja.
- Rentan halusinasi untuk fakta terbaru atau spesifik.
- Tidak bisa menghitung angka besar dengan tepat.

ReAct

- LLM bisa **memanggil tools** (search, calculator, database).
- Jawaban di-*ground* pada data nyata dari tool.
- Transparansi: kita bisa melihat "jejak pikiran" dan aksi yang diambil.

Insight

ReAct = Chain-of-Thought + Tool Use. Ini adalah fondasi dari **AI Agents** modern.

Apa Itu AI Agent?

- **AI Agent** = LLM yang diperlengkapi dengan kemampuan untuk bertindak secara otonom.
- Komponen utama AI Agent:

Komponen	Fungsi
LLM (Otak)	Memahami instruksi, mengambil keputusan
Tools (Tangan)	Search, calculator, code interpreter, API
Memory (Ingatan)	Menyimpan konteks percakapan & hasil sebelumnya
Planning (Strategi)	Memecah tugas besar menjadi sub-tugas

Analogi

LLM biasa = mahasiswa yang hanya menjawab dari kepala.
AI Agent = mahasiswa yang boleh buka buku, pakai kalkulator, dan browsing.

Function Calling: Cara LLM Memanggil Tools

- **Function Calling** = mekanisme terstruktur di mana LLM menghasilkan output berupa pemanggilan fungsi (bukan teks biasa).
- LLM tidak benar-benar *menjalankan* kode – dia hanya menghasilkan JSON yang mendeskripsikan fungsi mana yang dipanggil dan dengan parameter apa.

Contoh Output Function Calling

```
function": "get_weather","arguments" : "city" : "Bogor","date" : "2025 − 06 − 15"
Sistem menjalankan fungsi, lalu mengirim hasilnya kembali ke LLM: -> "Cuaca di Bogor: 28C, berawan" -> LLM merangkai jawaban final
untuk user
```

Arsitektur Agent: ReAct vs Plan-and-Execute

ReAct Agent

- Berpikir dan bertindak **satu langkah pada satu waktu**.
- Cocok untuk tugas yang tidak bisa diprediksi arahnya.
- Loop: Thought $\rightarrow$ Action $\rightarrow$ Observation.
- Kelemahan: bisa "tersesat" di loop panjang.

Plan-and-Execute Agent

- **Langkah 1:** Buat rencana lengkap (daftar sub-tugas).
- **Langkah 2:** Eksekusi tiap sub-tugas satu per satu.
- Cocok untuk tugas kompleks yang terstruktur.
- Kelemahan: rencana awal bisa salah.

Tools yang Umum Digunakan

Web Search, Calculator, Code Interpreter (Python REPL), Database Query (SQL), API Eksternal (cuaca, berita), File Reader (PDF, CSV).

Membuat Custom Tool di LangChain

Membuat Tool Sendiri

```
@tool def hitung_pk(nilai_list: list[float]) -> float: """Menghitung IPK mahasiswa dari daftar nilai matakuliah."""return sum(nilai_list)/len(nilai_list)
@tool def cari_jadwal(hari: str) -> str: """Mencari jadwal kuliah berdasarkan hari."""jadwal =
"senin": "08:00 - Kecerdasan Buatan (Lab 3)","selasa": "10:00 - Struktur Data (R.201)","rabu": "13:00 - Basis Data (Lab 2)",return jadwal.get(hari.lower(),"
tools = [hitung_pk, cari_jadwal]
```

Menjalankan Agent dengan Tools

LangChain Agent dengan Tools

```
1. Inisialisasi LLM llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
2. Ambil prompt template ReAct dari LangChain Hub prompt = hub.pull("hwchase17/react")
3. Buat agent agent = create_react_agent(llm = llm, tools = tools, prompt = prompt)
4. Buat executor (yang menjalankan loop Thought-Action-Observation)
agent_executor = AgentExecutor(agent = agent, tools = tools, verbose = True)
5. Jalankan result = agent_executor.invoke("input" : "BerapaIPKsayajikanilaiku[3.5, 4.0, 3.0, 3.7]?")print(result["output"])− > "IPKAndaadalah3.55"
```

Memory: Mengingat Percakapan

- Tanpa memory, setiap pemanggilan agent adalah percakapan baru (lupa segalanya).
- LangChain menyediakan beberapa jenis memory:

Jenis Memory	**Cara Kerja**
`ConversationBufferMemory`	Menyimpan seluruh riwayat chat apa adanya. Sederhana, tapi boros token.
`ConversationSummaryMemory`	Merangkum percakapan lama menggunakan LLM. Hemat token, tapi ada risiko kehilangan detail.
`ConversationBufferWindowMemory`	Hanya menyimpan k pesan terakhir (sliding window).

Contoh Penggunaan Memory

```
memory = ConversationBufferMemory(memory_key="chat_history", return_messages=True)Memoryotomatismenyimpanpesanuser AIdisetiapinteraksi
```

Studi Kasus 1: Chatbot FAQ Kampus STMIK Tazkia

Skenario:

- Mahasiswa sering bertanya hal yang sama: jadwal, biaya kuliah, prosedur KRS.
- Staff akademik kewalahan menjawab pertanyaan berulang.

Solusi AI Agent:

- **Tools:** RAG (dokumen panduan akademik) + Database query (jadwal, nilai).
- **Memory:** Menyimpan konteks mahasiswa (nama, NIM, prodi).
- **System Prompt:** "Kamu adalah asisten akademik STMIK Tazkia. Jawab berdasarkan dokumen resmi."

Contoh Interaksi

User: "Kapan batas pengisian KRS semester ini?"

Thought: Saya perlu mencari info batas KRS di dokumen akademik.

Action: `search_docs(''batas KRS'')`

Observation: "Batas KRS: 15 Sept 2025"

Answer: "Batas pengisian KRS semester ini adalah 15 September 2025."

Studi Kasus 2 & 3

Asisten Riset

- **Masalah:** Mahasiswa kesulitan mencari dan merangkum paper ilmiah.
- **Tools:**
  - `search_arxiv()` – cari paper
  - `summarize_pdf()` – ringkas paper
  - `translate()` – terjemahkan ke Bahasa Indonesia
- **Output:** Ringkasan paper + poin-poin kunci + referensi terstruktur (BibTeX).

UMKM Assistant

- **Masalah:** UMKM butuh rekomendasi produk untuk pelanggan.
- **Tools:**
  - `search_products()` – cari katalog
  - `check_stock()` – cek stok
  - `calculate_price()` – hitung diskon
- **Contoh:** "Saya cari batik untuk acara formal, budget 300rb" $\rightarrow$ Agent mencari, filter harga, cek stok, dan merekomendasikan.

Kesimpulan

1. **Prompt Engineering** adalah keterampilan kritis: zero-shot, few-shot, system prompt, dan pengaturan parameter decoding.
2. **Chain-of-Thought** memaksa LLM "berpikir" langkah demi langkah, meningkatkan akurasi penalaran.
3. **Self-Consistency** meningkatkan reliabilitas dengan majority voting dari multiple reasoning paths.
4. **ReAct** menggabungkan reasoning dengan acting, memungkinkan LLM mengakses dunia nyata via tools.
5. **AI Agents** = LLM + Tools + Memory + Planning – paradigma baru membangun aplikasi AI.
6. **LangChain** menyediakan framework untuk membangun agent dengan mudah.

Referensi

- Wei, J., et al. (2022). "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." *NeurIPS 2022*.
- Wang, X., et al. (2022). "Self-Consistency Improves Chain of Thought Reasoning in Language Models." *ICLR 2023*.
- Yao, S., et al. (2022). "ReAct: Synergizing Reasoning and Acting in Language Models." *ICLR 2023*.
- Brown, T., et al. (2020). "Language Models are Few-Shot Learners." *NeurIPS 2020*.
- LangChain Documentation: `https://docs.langchain.com`
- OpenAI Function Calling Guide: `https://platform.openai.com/docs/guides/function-calling`