ปั้น AI Agent ด้วย ADK

คู่มือสร้างเอเจนต์จากพื้นฐาน สู่การใช้งานจริง

อนุชิต ชโลธร

สำนักพิมพ์ก๊อปวาง

ปั้น AI Agent ด้วย ADK

คู่มือสร้างเอเจนต์จากพื้นฐานสู่การใช้งานจริง

อนุชิต ชโลธร

คำนำสำนักพิมพ์

ในยุคที่เทคโนโลยี AI ก้าวหน้าอย่างรวดเร็ว สำนักพิมพ์ของเรามีความยินดีเป็นอย่างยิ่งที่ได้นำเสนอหนังสือ “ปั้น AI Agent ด้วย ADK” หนังสือเล่มนี้เป็นกุญแจสำคัญที่จะเปิดประตูสู่โลกของ เอไอเอเจนต์ ซึ่งเป็นนวัตกรรมที่จะพลิกโฉมการทำงานและชีวิตประจำวันของเรา

เรามุ่งมั่นที่จะส่งมอบความรู้ที่เข้าถึงได้และนำไปใช้ได้จริง หนังสือเล่มนี้จึงถูกออกแบบมาเพื่อนักพัฒนาและผู้สนใจทุกระดับ ให้สามารถเรียนรู้การสร้างเอไอเอเจนต์ ตั้งแต่พื้นฐานจนถึงการประยุกต์ใช้ในสถานการณ์จริง ด้วย Agent Development Kit (ADK) ที่จะช่วยให้กระบวนการพัฒนามีประสิทธิภาพ

ขอขอบคุณผู้เขียนสำหรับความทุ่มเทในการสร้างสรรค์ผลงานอันทรงคุณค่านี้ เราหวังเป็นอย่างยิ่งว่า “ปั้น AI Agent ด้วย ADK” จะเป็นแรงบันดาลใจและเครื่องมือสำคัญในการขับเคลื่อนนวัตกรรม AI ของประเทศไทย

คำนำผู้เขียน

ปัญญาประดิษฐ์ (AI) ได้เข้ามาเป็นส่วนหนึ่งในชีวิตประจำวันของเรา และกำลังนำเราเข้าสู่ยุคใหม่ที่น่าตื่นเต้นยิ่งกว่า นั่นคือยุคของ **เอไอเอเจนต์ (AI Agent)** เอไอเอเจนต์ไม่ใช่เพียงโปรแกรม แต่เป็นระบบอัจฉริยะที่สามารถรับรู้, คิดวิเคราะห์ และตัดสินใจเพื่อบรรลุเป้าหมายได้อย่างอิสระ พวกมันคืออนาคตของซอฟต์แวร์ที่จะปฏิวัติวิธีที่เราทำงานและมีปฏิสัมพันธ์กับเทคโนโลยี

หนังสือ **“ปั้น AI Agent ด้วย ADK”** เล่มนี้ ถูกเขียนขึ้นเพื่อเป็นคู่มือสำหรับทุกคนที่ต้องการสำรวจและสร้างสรรค์ในโลกของเอไอเอเจนต์ ไม่ว่าคุณจะเป็นนักพัฒนาที่ต้องการเครื่องมืออันทรงพลัง หรือผู้ที่สนใจในเทคโนโลยี AI หนังสือเล่มนี้จะนำคุณไปทีละขั้นตอน

เราจะพาคุณเจาะลึก Agent Development Kit (ADK) ซึ่งเป็นเฟรมเวิร์กที่ออกแบบมาเพื่อทำให้การสร้างเอไอเอเจนต์เป็นเรื่องง่ายและมีประสิทธิภาพ คุณจะได้เรียนรู้ผ่านตัวอย่างโค้ดและกรณีศึกษาจริง ตั้งแต่การเชื่อมต่อกับโมเดลภาษาขนาดใหญ่ (LLM) การสร้างเครื่องมือ การออกแบบเวิร์กโฟลว์ การสร้างระบบอัลติเอเจนต์ ไปจนถึงการดีพลอย และการดูแลรักษาเอไอเอเจนต์ของคุณ

เป้าหมายของเราคือการทำให้หนังสือเล่มนี้เป็น “คู่มือฉบับลงมือทำ” ที่คุณสามารถนำไปใช้สร้างโปรเจกต์ของคุณได้ทันที เราหวังว่าหนังสือ

เล่มนี้จะเป็นจุดเริ่มต้นที่มั่นคงและเป็นแรงบันดาลใจให้คุณสร้างสรรค์เอไอเอเจนต์ที่น่าทึ่งต่อไป

ยินดีต้อนรับสู่โลกแห่งเอไอเอเจนต์ ขอให้สนุกกับการสร้างสรรค์อนาคตด้วยมือของคุณเอง

สารบัญ

แนะนำเอไอเอเจนต์

เอไอเอเจนต์ (AI Agent) หรือ **Agentic AI** คือระบบปัญญาประดิษฐ์ที่สามารถ *ตัดสินใจ วางแผน และลงมือทำงาน* ได้ด้วยตัวเอง เพื่อบรรลุเป้าหมายที่ได้รับมอบหมาย โดยไม่ต้องรอคำสั่งจากมนุษย์ในทุกขั้นตอน ทำให้ AI พัฒนาไปไกลกว่าเดิม จากการเป็นเพียง *ระบบตอบคำถาม* กลายเป็น *ผู้ช่วยอัจฉริยะ* ที่ลงมือทำงานแทนเราได้จริง

เบื้องหลังความสามารถนี้คือพลังของ **โมเดลภาษาขนาดใหญ่ (LLM)** และเฟรมเวิร์กสำหรับสร้างเอเจนต์ เช่น LangChain, LangGraph, CrewAI และ ADK ซึ่งช่วยให้นักพัฒนาสร้างเอเจนต์ที่ยืดหยุ่นและปรับให้เหมาะกับงานของตัวเองได้ง่ายขึ้น

คุณสมบัติหลักของเอไอเอเจนต์

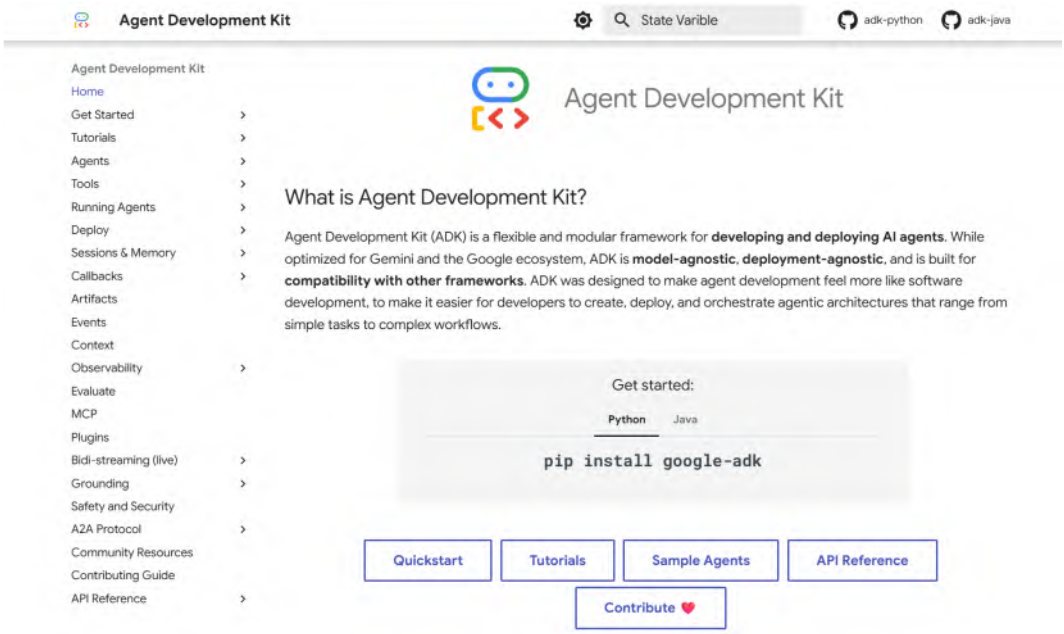
- **การตั้งเป้าหมาย (Goal Setting):** ผู้ใช้ระบุเป้าหมาย แล้วเอเจนต์ประเมินบริบทและข้อมูลเพื่อหาวิธีที่เหมาะสมที่สุด
- **การวางแผน (Planning):** แบ่งงานออกเป็นขั้นตอนย่อย วางลำดับงานอย่างเป็นระบบ
- **การใช้งานเครื่องมือ (Tool Usage):** เรียกใช้ API, ฐานข้อมูล และบริการภายนอกได้อย่างหลากหลาย
- **ความสามารถในการจดจำ (Memory):** บางระบบมีหน่วยความจำเพื่อเก็บบริบทและประสบการณ์ เพิ่มคุณภาพการตัดสินใจ

- **การทำงานอัตโนมัติหลายขั้นตอน (Autonomous Execution):** ทำงานต่อเนื่องด้วยตัวเอง ไม่ต้องคอยสั่งซ้ำ

ตัวอย่างเช่น ตั้งเอเจนต์ให้ **ค้นหาข่าวเทคโนโลยีทุกเช้า สรุป และส่งอีเมลอัตโนมัติ**—เอเจนต์สามารถจัดการทุกอย่างตั้งแต่ต้นจนจบ และทำซ้ำทุกวันโดยไม่ต้องตั้งค่าใหม่

Agent Development Kit (ADK) คืออะไร?

Agent Development Kit (ADK) คือเฟรมเวิร์กแบบโมดูลาร์สำหรับสร้างและดีพลอยเอไอเอเจนต์ ได้อย่างยืดหยุ่นและมีมาตรฐาน ADK ถูกออกแบบมาให้ทำงานได้อย่างราบรื่นในระบบนิเวศของ **Google และ Gemini** แต่ในทางปฏิบัติ *ไม่ได้ผูกติดกับแพลตฟอร์มหรือโมเดลใดเป็นพิเศษ* จึงสามารถนำไปใช้ร่วมกับเฟรมเวิร์กเอเจนต์อื่นๆ ได้เช่นกัน



google.github.io/adk-docs

แนวคิดของ ADK คือ “สร้างเอเจนต์ให้เหมือนพัฒนาซอฟต์แวร์จริง” — มีโครงสร้าง, ระบบทดสอบ, เครื่องมือ และแนวปฏิบัติที่ช่วยให้คุณสร้างเอเจนต์ได้ตั้งแต่งานง่ายๆ ไปจนถึงเวิร์กโฟลว์ที่ซับซ้อนระดับโปรดักชัน

จุดเด่นของ ADK

- **ระบบจัดการเวิร์กโฟลว์ที่ยืดหยุ่น:** รองรับการทำงานหลายรูปแบบ เช่น ทำงานทีละขั้น (Sequential), ทำงานพร้อมกัน (Parallel), ทำซ้ำแบบลูป (Loop) หรือให้ LLM ตัดสินใจเลือกเส้นทางของเวิร์กโฟลว์
- **สถาปัตยกรรมมัลติเอเจนต์:** เปิดโอกาสให้เอเจนต์หลายตัวที่เชี่ยวชาญต่างกันทำงานร่วมกัน เพื่อรองรับงานที่ซับซ้อนและ

ขยายระบบได้ง่าย

- **ระบบเครื่องมือที่หลากหลาย:** ใช้เครื่องมือสำเร็จรูป เชื่อมต่อไลบรารีภายนอก (เช่น LangChain, CrewAI) สร้างเครื่องมือใหม่ หรือแม้แต่ใช้เอเจนต์ตัวอื่นเป็นเครื่องมือได้
- **พร้อมดีพลอยในทุกสภาพแวดล้อม:** ใช้งานได้ตั้งแต่ในเครื่อง คอนเทนเนอร์ ไปจนถึงระบบคลาวด์ เช่น Vertex AI Agent Engine, Cloud Run หรือโครงสร้างพื้นฐานขององค์กร
- **มีระบบประเมินผลในตัว:** สามารถทดสอบคุณภาพของงานและขั้นตอนการทำงานด้วย Test Case ที่กำหนดได้ล่วงหน้า
- **เน้นความปลอดภัยและความน่าเชื่อถือ:** มีแนวทางด้านความปลอดภัยให้ปฏิบัติตามตั้งแต่ขั้นตอนออกแบบ เพื่อสร้างเอเจนต์ที่ใช้งานได้จริงอย่างมั่นใจ

ตัวอย่างการใช้งาน ADK

- **AI Assistant ส่วนตัว:** จัดการ Calendar, ส่งอีเมล, ค้นหาเอกสารใน Google Drive
- **Customer Support Agent:** เชื่อมต่อ CRM ดึงข้อมูลคำสั่งซื้อ และช่วยตอบคำถามอัตโนมัติ
- **DevOps Agent:** ตรวจสอบสถานะ Build/Deploy ผ่าน GitHub Actions หรือ CI/CD อื่นๆ

เครื่องมือสำหรับสร้างเอเจนต์ใน ADK

ADK มาพร้อมเครื่องมือครบชุดเพื่อรองรับนักพัฒนาทุกระดับ

- **ADK CLI & Library:** สำหรับเขียนโค้ดสร้างเอเจนต์โดยตรง
- **Agent Config:** สร้างเอเจนต์แบบ *ไม่ต้องเขียนโค้ด* เพียงกำหนดไฟล์ YAML
- **Visual Builder:** เครื่องมือแบบแผนภาพ (visual workflow) สำหรับออกแบบเวิร์กโฟลว์เอเจนต์อย่างรวดเร็ว

ด้วยชุดเครื่องมือเหล่านี้ ADK ช่วยให้นักพัฒนาก้าวข้ามการสร้าง AI แบบ Chatbot ไปสู่การสร้าง *เอเจนต์ที่ลงมือทำงานแทนมนุษย์* มีความสามารถในการจดจำ เชื่อมต่อระบบภายนอก ทำงานอัตโนมัติหลายขั้นตอน และพร้อมใช้งานจริงในระดับโปรดักชัน — เป็นเทคโนโลยีสำคัญที่นักพัฒนายุค AI ไม่ควรมองข้าม

สร้างเอเจนต์ด้วย ADK

LLM Agent หรือที่เราเรียก “Agent” เป็นส่วนประกอบหลักทำหน้าที่เป็นส่วน “การคิด” เราสามารถใช้ประโยชน์และความสามารถของ LLM ในการใช้เหตุผล การเข้าใจภาษาธรรมชาติ การตัดสินใจ การตอบสนองและการโต้ตอบกับเครื่องมือ จะแตกต่างจาก Workflow เพราะไม่มีการกำหนดขั้นตอนการทำงานไว้ล่วงหน้า เอเจนต์จะตีความคำสั่งจาก context และตัดสินใจดำเนินการด้วยตัวเองว่าจะทำงานอย่างไร จะเลือกใช้เครื่องมือใด หรือเรียกใช้เอเจนต์ตัวอื่นเพื่อดำเนินการหรือไม่

กำหนดตัวตนและวัตถุประสงค์ของเอเจนต์

ก่อนสร้างเอเจนต์ คุณควรกำหนดให้ชัดเจนว่าเอเจนต์นี้เป็นใครและมีหน้าที่อะไร การตั้งค่าเหล่านี้จะช่วยให้ทั้งคนและเอเจนต์อื่นๆ เข้าใจบทบาทของมันได้อย่างถูกต้อง

- **Name:** ชื่อเอเจนต์เป็นองค์ประกอบสำคัญสำหรับการอ้างอิงภายในระบบ ควรตั้งชื่อที่สื่อถึงหน้าที่ เช่น customer_support_router หรือ billing_inquiry_agent เพื่อให้เข้าใจบทบาทของเอเจนต์ได้ทันที
- **Description:** อธิบายความสามารถของเอเจนต์แบบกระชับและชัดเจน เพื่อให้เอเจนต์อื่นตัดสินใจได้ว่าควรส่งงานมาที่เอเจนต์นี้

หรือไม่ ควรระบุให้เฉพาะเจาะจง เช่น “Handles inquiries about current billing statements” มากกว่าคำกว้างๆ อย่าง “Billing agent”

- **Model:** ระบุโมเดล LLM ที่ใช้เป็นสมองของเอเจนต์ โดยเลือกโมเดลที่เหมาะกับระดับการใช้เหตุผล ความเร็ว และต้นทุน โมเดลที่ซับซ้อนขึ้นอาจให้ผลลัพธ์ดีขึ้น แต่แลกกับค่าใช้จ่ายและเวลาในการประมวลผลที่สูงขึ้น ดังนั้นควรเลือกให้สอดคล้องกับงานของเอเจนต์

ตัวอย่างการกำหนดตัวตนและวัตถุประสงค์ของเอเจนต์

```
# ตัวอย่าง: การกำหนดตัวตนพื้นฐาน
capital_agent = LlmAgent(
    model="gemini-2.5-flash",
    name="capital_agent",
    description="Answers user questions about the capital city of a given country."
    # กำหนดขั้นตอนการทำงานในขั้นตอนต่อไป
)
```

กำหนดขั้นตอนการทำงานของเอเจนต์

การกำหนดขั้นตอนการทำงาน (Instruction) มีความสำคัญมาก เนื่องจากการกำหนดลักษณะการทำงานของเอเจนต์ เช่น

- ภารกิจหลักและเป้าหมาย

- บุคลิกภาพหรือตัวตน เช่น คุณเป็นผู้ช่วยที่มีประโยชน์, คุณเป็นโจรสลัดที่ฉลาด เป็นต้น
- ข้อจำกัดเกี่ยวกับพฤติกรรม เช่น ตอบเฉพาะคำถามเกี่ยวกับ X อย่าเปิดเผย Y เป็นต้น
- วิธีการใช้เครื่องมือ

คุณควรอธิบายวัตถุประสงค์ของเครื่องมือแต่ละชิ้นและสถานการณ์ที่ควรเรียกใช้งานเครื่องมือนั้นๆ พร้อมทั้งอธิบายเพิ่มเติมเกี่ยวกับคำอธิบายภายในเครื่องมือนั้นๆ รวมไปถึงรูปแบบที่ต้องการสำหรับเอาต์พุต เช่น ตอบกลับในรูปแบบ JSON จัดทำรายการแบบหัวข้อย่อย เป็นต้น

ตัวอย่างการกำหนดค่า Instruction เพื่อกำหนดพฤติกรรมของเอเจนต์อย่างละเอียด

```
# ขั้นตอนการทำงาน
agent_instruction = """You are an agent that provides the
capital city of a country.
When a user asks for the capital of a country:
1. Identify the country name from the user's query.
2. Use the `get_capital_city` tool to find the capital.
3. Respond clearly to the user, stating the capital city.
Example Query: "What's the capital of France?"
Example Response: "The capital of France is Paris."
"""

capital_agent = LlmAgent(
```

```
    model="gemini-2.5-flash",
    name="capital_agent",
    description="Answers user questions about the capital
city of a given country.",
    instruction=agent_instruction,
    # กำหนดเครื่องมือในขั้นตอนต่อไป
)
```

อธิบาย Instruction ข้างต้น

- **บทบาท (Role):** “คุณคือเอเจนต์ที่ให้ข้อมูลเมืองหลวงของประเทศ” (You are an agent that provides the capital city of a country.)
- **กระบวนการทำงาน (Process):** อธิบายขั้นตอนที่ต้องทำเมื่อผู้ใช้ถามหาเมืองหลวง
    - ระบุชื่อประเทศจากคำถามของผู้ใช้ (Identify the country name from the user’s query.)
    - ใช้เครื่องมือ (tool) ที่ชื่อ get_capital_city เพื่อค้นหาเมืองหลวง (Use the get_capital_city tool to find the capital.)
    - ตอบผู้ใช้อย่างชัดเจน โดยระบุชื่อเมืองหลวง (Respond clearly to the user, stating the capital city.)
- **ให้ตัวอย่าง (Examples):** ให้ตัวอย่างคำถามและคำตอบที่คาดหวัง เพื่อให้ LLM เข้าใจรูปแบบที่ต้องการ เช่น ถาม “What’s the

capital of France?” ต้องตอบ “The capital of France is Paris.”

เคล็ดลับการสั่งงานเอเจนต์

เคล็ดลับการสั่งงานเอเจนต์ให้ได้ผลลัพธ์ที่ต้องการ

- **ชัดเจนและเฉพาะเจาะจง:** หลีกเลี่ยงความคลุมเครือ ระบุการดำเนินการและผลลัพธ์ที่ต้องการอย่างชัดเจน
- **ใช้ Markdown:** สำหรับคำแนะนำที่ซับซ้อนโดยใช้หัวเรื่อง รายการ เป็นต้น
- **ให้ตัวอย่าง (ตัวอย่างสั้นๆ):** สำหรับงานที่ซับซ้อนหรือรูปแบบเอาต์พุตเฉพาะ ให้รวมตัวอย่างในคำแนะนำ
- **ใส่คำแนะนำการใช้เครื่องมือ:** อย่าแสดงรายการเครื่องมืออย่างเดียว ให้อธิบายว่าเอเจนต์ควรใช้เครื่องมือเหล่านั้นตอนไหนและทำไม

การเรียกใช้เครื่องมือ

เครื่องมือ (Tool) มีบทบาทสำคัญในการเพิ่มขีดความสามารถให้กับเอเจนต์ เนื่องจากช่วยให้เอเจนต์สามารถเชื่อมต่อกับบริการภายนอกหรือระบบต่างๆ ได้ ไม่ว่าจะเป็นการคำนวณข้อมูล การดึงข้อมูลแบบเรียลไทม์ หรือการดำเนินการเฉพาะทางบางอย่างที่เกินขอบเขตของการใช้โมเดลภาษาเพียงอย่างเดียว

ภายในระบบของ ADK คุณสามารถระบุรายการเครื่องมือที่เอเจนต์สามารถใช้ได้ผ่านพารามิเตอร์ tools ซึ่งเครื่องมือแต่ละรายการสามารถอยู่ในรูปแบบต่างๆ ได้แก่ ฟังก์ชัน Python ที่ถูกห่อหุ้มให้อยู่ในรูปแบบ FunctionTool โดยอัตโนมัติ หรือคลาสที่สืบทอดจาก BaseTool ซึ่งในกรณีนี้อาจเป็นอินสแตนซ์ของเอเจนต์ตัวอื่น โดยจะใช้ในลักษณะของ AgentTool ซึ่งเปิดโอกาสให้สามารถมอบหมายงานระหว่างเอเจนต์ได้

เมื่อมีการสนทนาเกิดขึ้น โมเดลภาษา (LLM) จะใช้ข้อมูลจากชื่อฟังก์ชันหรือชื่อของเครื่องมือ คำอธิบายที่มาจาก docstring หรือฟิลด์ description รวมถึงโครงร่างของพารามิเตอร์ (parameter schema) เพื่อวิเคราะห์และตัดสินใจว่าในสถานการณ์นั้นควรเรียกใช้เครื่องมือใด พร้อมทั้งจัดเตรียมอินพุตให้เหมาะสมสำหรับการเรียกใช้งานเครื่องมือนั้น

ตัวอย่างการกำหนดเครื่องมือ get_capital_city ให้เอเจนต์

```
# กำหนดฟังก์ชันเครื่องมือ
def get_capital_city(country: str) -> str:
  """Retrieves the capital city for a given country."""
  # ส่วนนี้ให้คุณแทนที่ด้วยตรรกะจริง เช่น การเรียก API, การค้นหาฐานข้อมูล เป็นต้น
  capitals = {"france": "Paris", "japan": "Tokyo", "canada": "Ottawa"}
  return capitals.get(country.lower(), f"Sorry, I don't know the capital of {country}.")
```

```
# ขั้นตอนการทำงาน
agent_instruction = """You are an agent that provides the
capital city of a country.
When a user asks for the capital of a country:
1. Identify the country name from the user's query.
2. Use the `get_capital_city` tool to find the capital.
3. Respond clearly to the user, stating the capital city.
Example Query: "What's the capital of France?"
Example Response: "The capital of France is Paris."
"""


# เพิ่มเครื่องมือให้เอเจนต์
capital_agent = LlmAgent(
    model="gemini-2.5-flash",
    name="capital_agent",
    description="Answers user questions about the capital
city of a given country.",
    instruction=agent_instruction,
    tools=[get_capital_city] # ใส่ชื่อฟังก์ชั่นไปตรงๆ
)
```

การจัดการตัวแปรสถานะ

การจัดการตัวแปรสถานะ (State Variable) ช่วยให้เอเจนต์สามารถนำค่าที่กำหนดไว้มาใช้ซ้ำหรือแทรกลงในคำสั่งได้อย่างยืดหยุ่น

หลักการใช้งาน

- คำสั่งจะอยู่ในรูป *Template String* ซึ่งสามารถใช้ {var} เพื่อแทรกค่าตัวแปรแบบไดนามิก
- {var} หมายถึงการเรียกค่าของตัวแปรสถานะที่ชื่อว่า var
- {artifact.var} หมายถึงการนำข้อความจาก *artifact* ที่ชื่อว่า var มาใช้
- หากไม่มีตัวแปร state หรือ artifact ที่อ้างถึง เอเจนต์จะแสดงข้อผิดพลาด แต่หากต้องการไม่ให้ระบบหยุดทำงาน สามารถใส่ ? ต่อท้ายชื่อตัวแปรได้ เช่น {var?}

การเรียกใช้เครื่องมือเป็นแกนหลักที่ทำให้เอเจนต์ก้าวข้ามขีดจำกัดของโมเดลภาษา สามารถเชื่อมต่อข้อมูลภายนอก ประมวลผลเชิงตรรกะ และลงมือทำงานที่ซับซ้อนได้จริง ผ่านการกำหนดรายการเครื่องมือในพารามิเตอร์ tools เอเจนต์จึงสามารถเลือกใช้ฟังก์ชันที่เหมาะสมตามบริบทของข้อความสนทนาได้โดยอัตโนมัติ โดยอาศัยทั้งชื่อเครื่องมือ คำอธิบาย และสคีมาของพารามิเตอร์เป็นตัวช่วยตัดสินใจ

แนวทางนี้ทำให้การออกแบบเอเจนต์มีความยืดหยุ่นสูง ไม่ว่าจะเป็นการใช้ฟังก์ชัน Python ธรรมดาเปลี่ยนให้เป็นเครื่องมือ โดยใช้ FunctionTool หรือการใช้อีกเอเจนต์หนึ่งเป็นเครื่องมือผ่าน AgentTool เพื่อสร้างการทำงานแบบมอบหมายข้ามเอเจนต์ เมื่อเข้าใจหลักการของเครื่องมือแล้ว ขั้นตอนต่อไปคือการนำองค์ประกอบทั้งหมดมาใช้ในการสร้างเอเจนต์ที่พร้อมทำงานจริงในระบบของคุณ

เริ่มสร้างเอเจนต์

เรามาสร้างเอเจนต์ พร้อมเรียนรู้วิธีใช้งานในหลายรูปแบบ ตั้งแต่การสร้างเอเจนต์ง่ายๆ ที่ไม่ต้องพึ่งพา Function Call หรือเครื่องมือใดๆ กับ Imagine Agent ไปจนถึงการเรียกใช้งานเครื่องมือผ่าน Function Tool กับ News Agent และการใช้เครื่องมือ build-in อย่าง Google Search กับ Search Agent เพื่อให้คุณเข้าใจและทดลองสร้างเอเจนต์แบบครบวงจร

ติดตั้ง ADK

ติดตั้ง ADK โดยใช้คำสั่ง

```
pip install google-adk
```

หรือใช้ uv ดังนี้

```
uv tool install google-adk
```

โครงสร้างโปรเจกต์เป็นดังนี้

```
adk-basic-agent/
├── imagine_agent/ # prompt refinement agent
│     ├── __init__.py
│     └── agent.py
├── search_agent/ # search tool agent
```

```
│     ├── __init__.py
│     └── agent.py
├── news_agent/  # news reseach agent
│     ├── __init__.py
│     └── agent.py
├── README.md
```

สร้าง Imagine Agent

สร้าง Imagine Agent ทำหน้าที่ปรับปรุง Prompt สำหรับสร้างรูปภาพของผู้ใช้ให้ดีขึ้น โดยเอเจนต์ใช้ความสามารถของ LLM ทำหน้าที่สร้าง Prompt ให้มีรายละเอียดมากขึ้น โดยไม่จำเป็นต้องมี Function Call หรือเครื่องมือใดๆ เพิ่มเติม เพียงใส่ขั้นตอนและวิธีการทำงานลงใน Instruction ของเอเจนต์เท่านั้น

สร้างเอเจนต์ imagine_agent ด้วยคำสั่ง

```
adk create imagine_agent
```

เลือก model ที่ต้องการใช้งาน ตัวอย่างนี้ใช้ model gemini-2.5-flash จาก Google AI และ กรอก Google API key ให้เรียบร้อย ในกรณีที่เลือกใช้ model จาก Vertex AI จะต้องตั้งค่า Google Cloud Project, Google Cloud Location ด้วย

```
Choose a model for the root agent:
1. gemini-2.5-flash
2. Other models (fill later)
Choose model (1, 2): 1
1. Google AI
2. Vertex AI
Choose a backend (1, 2): 1

Don't have API Key? Create one in AI Studio:
https://aistudio.google.com/apikey

Enter Google API key:
```

ADK จะสร้างไดเรคทอรี imagine_agent สร้างไฟล์ __init__.py, agent.py และ .env เพื่อเก็บ API KEY ดังนี้

```
GOOGLE_GENAI_USE_VERTEXAI=0
GOOGLE_API_KEY="YOUR_GEMINI_API_KEY"
```

แก้ไขไฟล์ agent.py ในไดเรคทอรี imagine_agent กำหนดตัวตนและขั้นตอนการทำงานของเอเจนต์ ดังนี้

ตัวอย่างโค้ด

```python
from google.adk.agents import Agent

# --- Define the Imagine Agent ---
imagine_agent_instruction = """
```

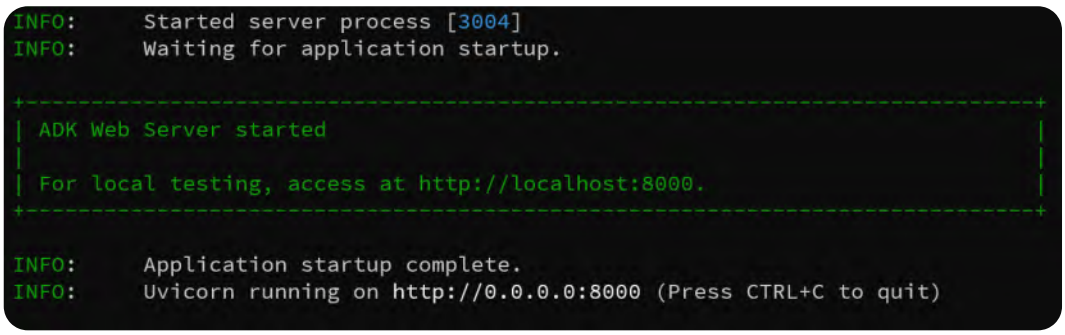
```
### **Optimized Instruction Prompt: Imagine Prompt Engineer**


You are an **Imagine Prompt Engineer**.
Your task is to transform any given topic, idea, or description into a **rich, detailed, and visually compelling prompt** suitable for an AI image generation model.


**Guidelines:**


* Identify the **main subject(s)** and **actions**.
* Describe the **setting**, **environment**, **background**, and **foreground** elements.
* Define the **mood**, **atmosphere**, and **lighting** (e.g., cinematic, soft, neon).
* Specify the **emotional tone** and overall **visual impression**.
* Optionally include **artistic or stylistic cues** (e.g., photorealistic, watercolor, cyberpunk, anime) — only when they enhance the concept or if requested.
* Integrate key **details**, **keywords**, and **concepts** from the user's input.
* Keep the result **concise** and **effective** — typically **1–3 sentences**.
* **CRITICAL:** Output **only** the final prompt text — no introductions, explanations, or extra commentary.


**Example:**

*Input:* `cat`
*Output:* `A fluffy ginger cat sleeping peacefully in a sunbeam on a wooden floor, photorealistic style.`
*Invalid Output:* `Here is the prompt: A fluffy ginger cat...`
"""
```

```
# Create the Imagine Agent instance
# It doesn't need specific tools for this core task,
# it relies on the LLM's text generation capability.
root_agent = Agent(
    name="imagine_agent",
    instruction=imagine_agent_instruction,
    # Standardize model for consistency within the flow
    model="gemini-2.5-flash"

)
```

เปิดไปที่ไดเรคทอรี adk-basic-agent และ สั่ง run web interface ด้วยคำสั่ง

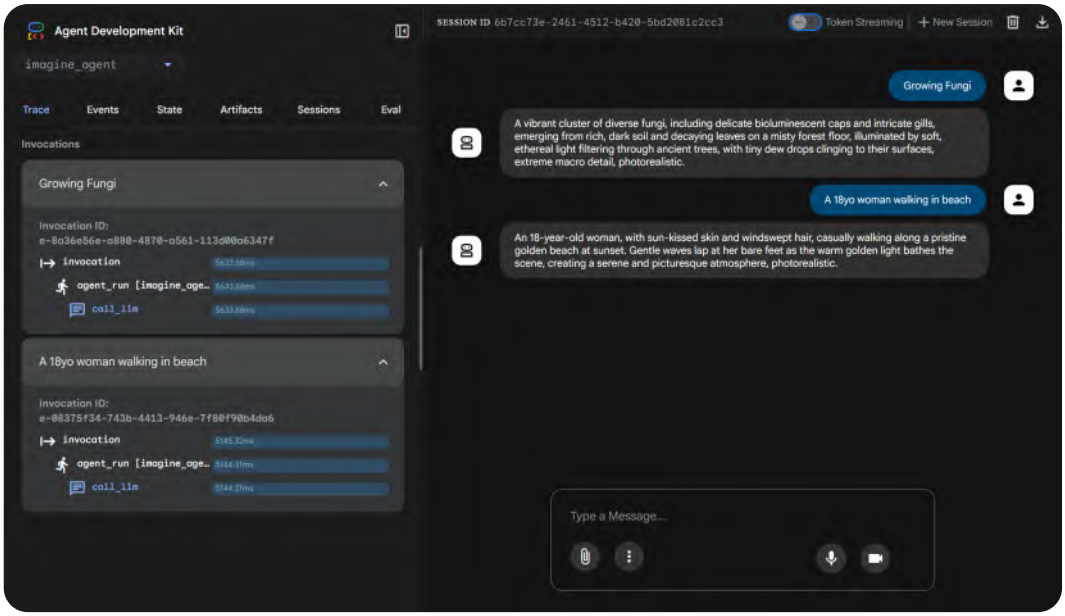
```
adk web
```

คุณจะพบว่า ADK Web Server ทำงานที่พอร์ต 8000 ดังภาพ



ADK Web Server ทำงานที่พอร์ต 8000

เปิดเบราเซอร์ไปที่ http://localhost:8000



ADK Dev UI

ลองใส่ prompt แบบกว้างๆ ลงไป เช่น

- “Glowing Fungi” คุณจะได้ prompt ที่ระบุรายละเอียดเพิ่มเติมกลับมาดังนี้

```
A vibrant cluster of diverse fungi, including delicate
bioluminescent caps and intricate gills, emerging from
rich, dark soil and decaying leaves on a misty forest
floor, illuminated by soft, ethereal light filtering
through ancient trees, with tiny dew drops clinging to
their surfaces, extreme macro detail, photorealistic.
```

- “A 18yo woman walking in beach” คุณจะได้ prompt ที่ระบุรายละเอียดเพิ่มเติมกลับมาดังนี้

```
An 18-year-old woman, with sun-kissed skin and windswept
hair, casually walking along a pristine golden beach at
sunset. Gentle waves lap at her bare feet as the warm
golden light bathes the scene, creating a serene and
picturesque atmosphere, photorealistic.
```

จากตัวอย่างนี้คุณสามารถใช้ prompt ที่ปรับปรุงแล้ว ไปใช้กับเครื่องมือ (tool) หรือ Agent อื่นเพื่อสร้างรูปภาพต่อไปได้

A vibrant cluster of diverse fungi, including delicate bioluminescent caps and intricate gills, emerging from rich, dark soil and decaying leaves on a misty forest floor, illuminated by soft, ethereal light filtering through ancient trees, with tiny dew drops clinging to their surfaces, extreme macro detail, photorealistic.

An 18-year-old woman, with sun-kissed skin and windswept hair, casually walking along a pristine golden beach at sunset. Gentle waves lap at her bare feet as the warm golden light bathes the scene, creating a serene and picturesque atmosphere, photorealistic.

สร้าง Search Agent เรียกใช้เครื่องมือภายใน ADK

เราสามารถเรียกใช้เครื่องมือ (tool) ผ่านทาง Function call ของ LLM ได้โดยใช้ internal tools ใน ADK ได้ เช่น Google Search, Code Excution, Vertex AI Search หรือใช้จากไลบรารีภายนอก เช่น LangChain หรือเรียกใช้ tool จาก MCP Server ได้เช่นกัน ADK ได้เตรียมเครื่องมือเหล่านี้ไว้ให้พร้อมใช้เรียบร้อยแล้ว ลองมาดูตัวอย่างโค้ด search_agent โดยใช้งานบริการ Google Search กัน

สร้างไดเรคทอรี search_agent สำหรับสร้างเอเจนต์สำหรับค้นหาข้อมูล

```
adk create search_agent
```

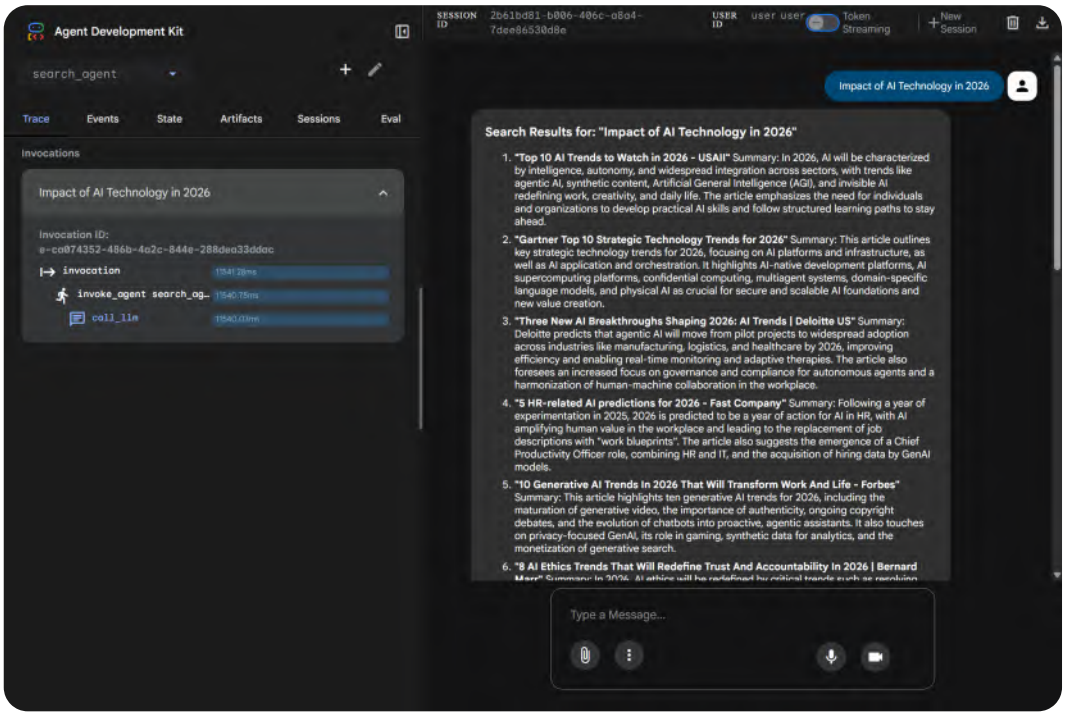
จากนั้นแก้ไขไฟล์ agent.py ลงในไดเรคทอรี search_agent กำหนดเอเจนต์และเครื่องมือ (tool) ที่ต้องใช้ดังนี้

```
from google.adk.agents import Agent
from google.adk.tools.google_search_tool import
GoogleSearchTool


search_agent_instruction = """
You are a helpful assistant.

Use the `GoogleSearchTool` to find relevant and up-to-date
information based on the user's query.
```

```
The tool will return a list of articles containing
**titles**, **summaries**.


Your task is to:


1. Review the list of returned articles.
2. Summarize the key information from these sources into a
concise report.
3. Present the findings in a **clear list format**,
including for each article:


   * **Title** – the article's name.
   * **Short Summary** – a brief 1–2 sentence overview of
its main points.


**Output Format Example:**


### Search Results for: *"Latest trends in AI agents
2025"*


1. **"The Rise of Agentic AI: How Autonomous Systems Are
Shaping 2025"**
   Summary: Discusses how AI agents are moving beyond
simple automation to handle complex workflows with
reasoning and tool use.


2. **"Google's Next-Gen Agent Framework: What Developers
Need to Know"**
   Summary: Covers Google's new platform for building
adaptive, multimodal AI agents with integrated tool
orchestration.


3. **"OpenAI and the Future of Autonomous Agents"**
   Summary: Explores OpenAI's approach to agent-native
architecture and the integration of external APIs and
reasoning modules.
```

```
"""

root_agent = Agent(
    name="search_agent",
    model="gemini-2.5-flash",
    description="Agent to search information",
    instruction=search_agent_instruction,
    tools=[GoogleSearchTool()]
)
```

จากโค้ดข้างต้นเพียงแค่กำหนด GoogleSearchTool ลงไปเท่านั้น คุณก็สามารถเชื่อมต่อและเรียกใช้งาน Google Search ได้เลย



ตัวอย่าง Search Agent

สร้าง News Agent เรียกใช้เครื่องมือภายนอก

ในส่วนนี้จะยกตัวอย่างการสร้าง AI Search Assistant ด้วย Agent Development Kit ร่วมกับ Gemini API เพื่อให้ระบบสามารถค้นหาข้อมูลและสรุปข้อมูลให้ผู้ใช้ได้โดยอัตโนมัติ ตัวอย่างนี้ใช้บริการค้นหาข้อมูลจาก Brave Search ซึ่งเป็นเครื่องมือจากภายนอก เราจะใช้ไลบรารีจาก LangChain ชื่อ BraveSearchWrapper เพื่อเรียกใช้งาน

สร้างไดเรคทอรี news_agent เพื่อสร้างเอเจนต์ค้นหาข้อมูล

```
adk create news_agent
```

แก้ไขไฟล์ .env ในไดเรคทอรี news_agent เพื่อเก็บ API KEY ของบริการต่างๆ

```
GOOGLE_GENAI_USE_VERTEXAI=0
GOOGLE_API_KEY="YOUR_GEMINI_API_KEY"
BRAVE_SEARCH_API_KEY="YOUR_BRAVE_SEARCH_API_KEY"
```

แก้ไขไฟล์ agent.py ในไดเรคทอรี news_agent กำหนด Agent และเครื่องมือ (tool) ที่ต้องใช้ดังนี้

```
from google.adk.agents import Agent
from langchain_community.utilities import
BraveSearchWrapper
import os

# Ensure you have BRAVE_SEARCH_API_KEY set in your
environment variables
```

```
if "BRAVE_SEARCH_API_KEY" not in os.environ:
    raise ValueError("BRAVE_SEARCH_API_KEY environment
variable not set.")



def get_news(query: str) -> dict:
    """
    Retrieve news articles including title, snippet, and
URL using Brave Search.

    Args:
        query (str) : Query the Brave search engine.

    Returns:
        dict: status and a formatted string containing
search results or an error message.
    """
    if query:
        print("-- brave search --")
        try:
            brave =
BraveSearchWrapper(api_key=os.environ["BRAVE_SEARCH_API_KE
Y"])
            results = brave.run(query)

            print(results)

        except Exception as e:
            return {"status": "error", "error_message":
f"Brave Search failed: {e}"}

        return {"status": "success", "result": results}
    else:
```

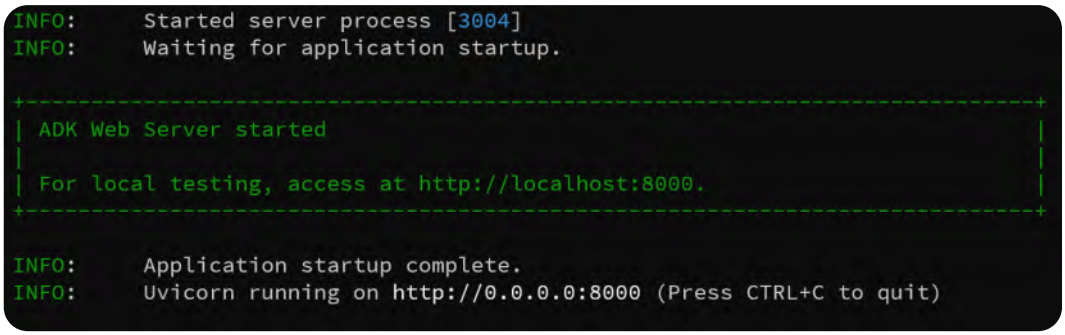
```
        return {"status": "error", "error_message": "topic
cannot be empty"}



news_search_instruction = """
**Role:**
You are a helpful research assistant.


**Goal:**
Use the `get_news` to find the most relevant and reliable
information based on the user's query.


**Capabilities:**


* You can access the `get_news`, which returns a list of
articles containing **titles**, **summaries**.
* You should read and interpret these results carefully.


**Instructions:**


1. Use the `get_news` to perform a search using the user's
query.
2. Review the list of returned articles.
3. For each article, extract:


   * **Title** – the name of the article or source.
   * **Short Summary** – a brief, 1–2 sentence summary of
the main idea or key takeaway.
4. Present your findings in a clear, user-friendly **list
format** as shown below.
5. If multiple perspectives exist, briefly highlight them.
6. Avoid redundant entries or unreliable sources.
7. Keep the final report concise, factual, and easy to
read.
```

```
**Output Format Example:**

### Search Results for: *"Latest trends in AI agents 2025"*

1. **"The Rise of Agentic AI: How Autonomous Systems Are Shaping 2025"**
   Summary: Discusses how AI agents are moving beyond simple automation to handle complex workflows with reasoning and tool use.

2. **"Google's Next-Gen Agent Framework: What Developers Need to Know"**
   Summary: Covers Google's new platform for building adaptive, multimodal AI agents with integrated tool orchestration.

3. **"OpenAI and the Future of Autonomous Agents"**
   Summary: Explores OpenAI's approach to agent-native architecture and the integration of external APIs and reasoning modules.
"""


root_agent = Agent(
    name="news_agent",
    model="gemini-2.5-flash",
    description="Agent to search news and information",
    instruction=news_search_instruction,
    tools=[get_news],
)
```

เปิดไปที่ไดเรคทอรี adk-basic-agent และ สั่ง run web interface ด้วยคำสั่ง

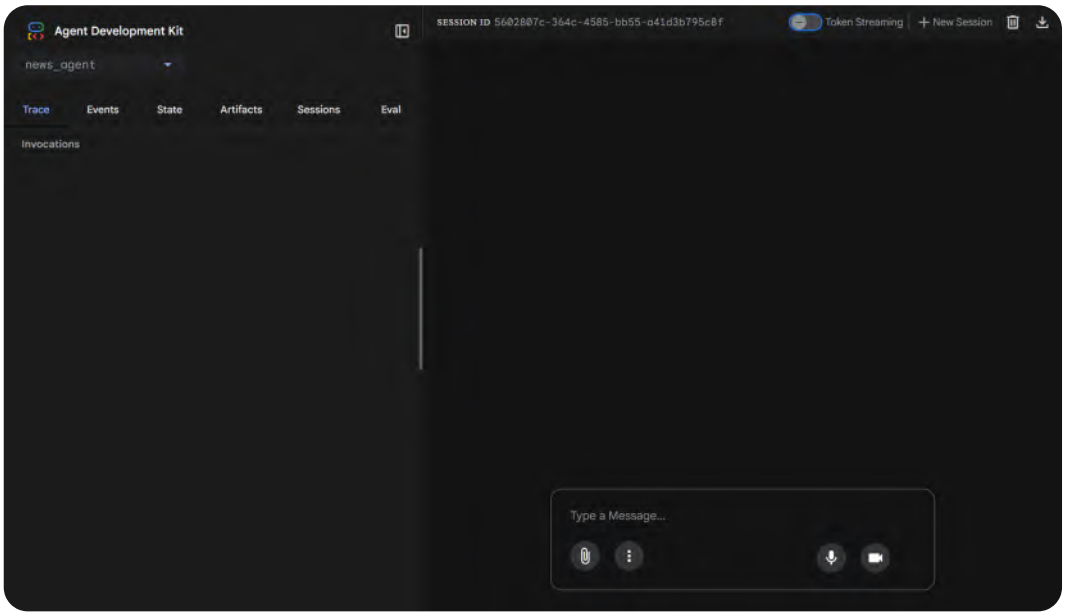
```
adk web
```

คุณจะพบว่า ADK Dev UI ทำงานที่พอร์ต 8000 ดังภาพ



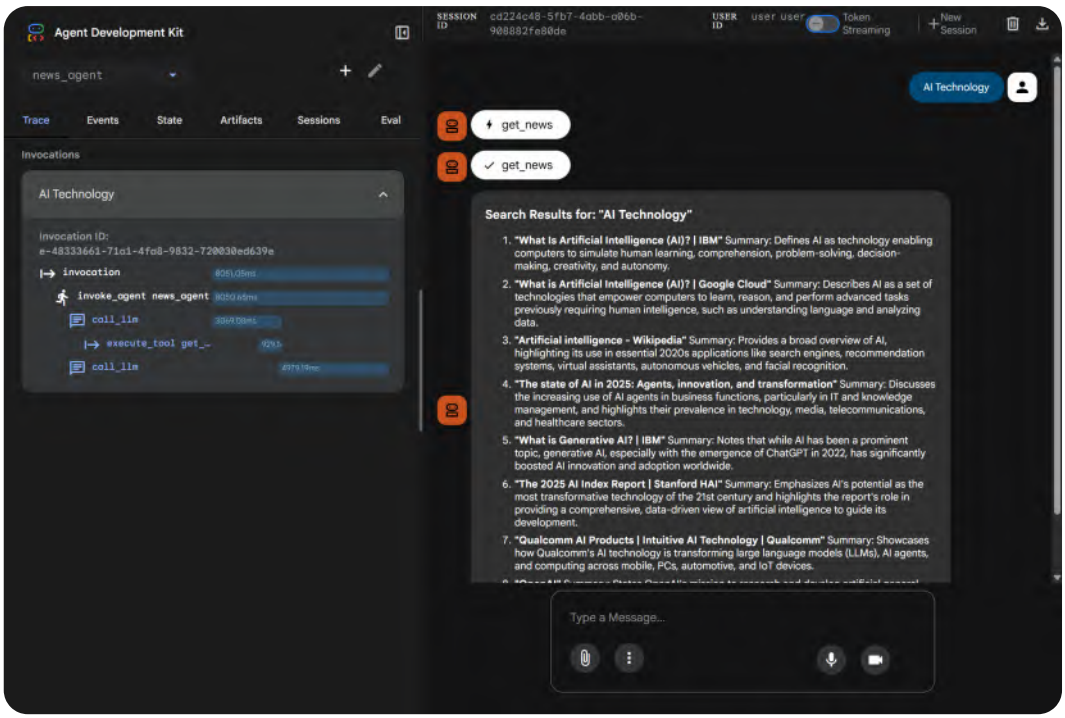
ADK Dev UI ทำงานที่พอร์ต 8000

เปิดเบราเซอร์ไปที่ http://localhost:8000 เลือก news_agent



ADK Dev UI

ทดลองกรอกข้อมูล “AI technology” คุณจะพบว่า Gemini เรียกเครื่องมือ get_news เพื่อค้นหาข่าวและสรุปข้อมูลข่าวดังภาพ



ค้นหาข่าวและสรุปข้อมูลข่าว

การรันแบบ ChatLoop ใน Command Line

คุณสามารถ run แบบ ChatLoop ใน Command Line ได้โดยใช้คำสั่ง run แล้วตามด้วยชื่อเอเจนต์ที่ต้องการ run ดังนี้

```
adk run news_agent
```

คุณจะพบว่าโปรแกรมจะมี prompt ให้แชทกับเอเจนต์ ให้ลองค้นหา “AI news” และดูผลลัพธ์ดังนี้

```
[user]: the impact of AI in 2026

-- brave search --
[{"title": "AI\u2019s true impact will become apparent in the coming year", "link": "https://www.economist.com/the-world-ahead/2025/11/10/ais-true-impact-will-become-apparent-in-the-coming-year", "snippet": "Yet revenues from AI so far amount ... feats of AI, the focus is shifting. In 2026 expect its <strong>economic, financial and social consequences to grab attention</strong>...."},...]

[news_agent]: ### Search Results for: "the impact of AI in 2026"

1.  **"AI’s true impact will become apparent in the coming year"**
    Summary: In 2026, the economic, financial, and social consequences of AI are expected to become much more evident.

2.  **"Top 10 AI Trends to Watch in 2026"**
    Summary: By 2026, AI is projected to significantly increase efficiency, improve patient outcomes, and reduce costs within modern healthcare systems.

3.  **"AI in 2026: The Breakthroughs, Challenges, and Real-World Impact Ahead"**
    Summary: AI's increasing sophistication and availability will change entire sectors, transforming
```

```
jobs, workflows, and bringing about new ethical
considerations.

    ...

[user]:
```

การรันแบบ API Server

คุณสามารถ run เอเจนต์ของคุณแบบ API Server ได้ โดยใช้คำสั่ง

```
adk api_server
```

API ทำงานที่ port 8000 หากต้องการ ส่ง HTTP request ไปยัง API Server ดังนี้

สร้าง Session สำหรับเอเจนต์โดยระบุ

- ชื่อ app เป็น imagin_agent
- User ID เป็น user_123
- Session ID เป็น s_123

ใช้คำสั่ง curl ดังนี้

```
curl -X POST
http://127.0.0.1:8000/apps/imagin_agent/users/u_123/sessio
ns/s_123
-H "Content-Type: application/json"
```

เซิร์ฟเวอร์จะสร้าง session ให้ดังนี้

```
{
  "id": "s_123",
  "appName": "imagin_agent",
  "userId": "u_123",
  "state": {},
  "events": [],
  "lastUpdateTime": 1759919980.1439106
}
```

ส่ง prompt ไปยัง Agent ดังนี้

```
curl -X POST http://127.0.0.1:8000/run
-H "Content-Type: application/json"
-d '{
  "app_name": "imagin_agent",
  "user_id": "u_123",
  "session_id": "s_123",
  "new_message": {
    "role": "user",
    "parts": [{
      "text": "Glowing fungi"
    }]
```

```
  }
}'
```

หากคุณใช้ /run คุณจะเห็นผลลัพธ์ของเหตุการณ์ทั้งหมดแสดงพร้อมกันในรูปแบบของรายการ (list) ซึ่งจะมีลักษณะคล้ายกับตัวอย่างต่อไปนี้

```
[
  {
    "content": {
      "parts": [
        {
          "text": "A magical forest floor at twilight,
covered in a vibrant array of bioluminescent mushrooms and
glowing moss, their soft, ethereal light illuminating dew-
kissed leaves, ancient roots, and a misty background, rich
cinematic lighting, hyperrealistic."
        }
      ],
      "role": "model"
    },
    "finishReason": "STOP",
    "usageMetadata": {
      "candidatesTokenCount": 47,
      "promptTokenCount": 301,
      "promptTokensDetails": [
        {
          "modality": "TEXT",
          "tokenCount": 301
        }
```

```
      ],
      "thoughtsTokenCount": 412,
      "totalTokenCount": 760
    },
    "invocationId": "e-1bf32d62-d10f-40b8-8738-
4b0cced92ac0",
    "author": "imagine_agent",
    "actions": {
      "stateDelta": {},
      "artifactDelta": {},
      "requestedAuthConfigs": {},
      "requestedToolConfirmations": {}
    },
    "id": "7dec21e0-a309-4d22-a8ea-50a4651f5393",
    "timestamp": 1759920038.216567
  }
]
```


หากคุณใช้ /run_sse

```
curl -X POST http://127.0.0.1:8000/run_sse
-H "Content-Type: application/json"
-d '{
  "app_name": "imagin_agent",
  "user_id": "u_123",
  "session_id": "s_123",
  "new_message": {
    "role": "user",
    "parts": [{
      "text": "Glowing fungi"
```

```
    }]
  }
  "streaming": true
}'
```

คุณสามารถตั้งค่า streaming เป็น true เพื่อเปิดใช้งานการสตรีมในระดับโทเค็น ซึ่งหมายความว่าผลลัพธ์จะถูกส่งกลับมาเป็น data หลายส่วน (chunks) ต่อเนื่องกัน ผลลัพธ์ที่คุณเห็นจะมีลักษณะคล้ายกับตัวอย่างต่อไปนี้


```
data: { "content": { "parts": [ { "text": "A hidden grotto
deep within an ancient forest, illuminated entirely by
clusters of various bioluminescent fungi casting a soft
emerald green and sapphire blue glow onto damp moss,
gnarled tree roots, and glistening droplets of water,
mystical atmosphere, hyperrealistic" } ], "role": "model"
}, "partial": true, "usageMetadata": {
"candidatesTokenCount": 49, "promptTokenCount": 507,
"promptTokensDetails": [ { "modality": "TEXT",
"tokenCount": 507 } ], "thoughtsTokenCount": 595,
"totalTokenCount": 1151 }, "invocationId": "e-7cfd3f16-
2522-4444-acc2-55afd9b5dc9f", "author": "imagine_agent",
"actions": { "stateDelta": {}, "artifactDelta": {},
"requestedAuthConfigs": {}, "requestedToolConfirmations":
{} }, "id": "9486d97a-397d-46de-8872-650121ca8e5a",
"timestamp": 1759920688.028734 }
```

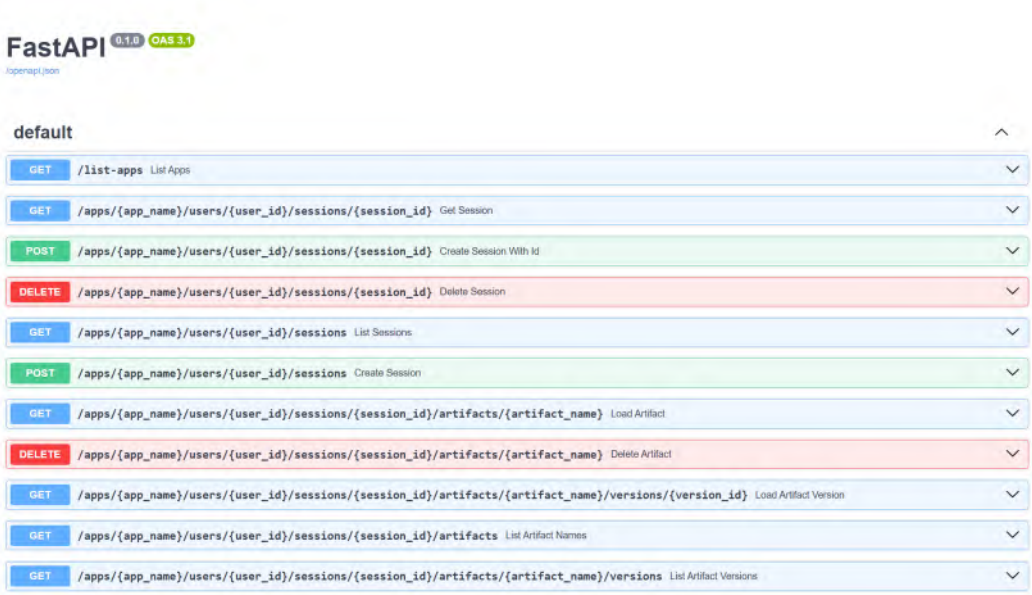
```
data: { "content": { "parts": [ { "text": ", cinematic
lighting, macro photography." } ], "role": "model" },
"partial": true, "finishReason": "STOP", "usageMetadata":
{ "candidatesTokenCount": 56, "promptTokenCount": 507,
"promptTokensDetails": [ { "modality": "TEXT",
"tokenCount": 507 } ], "thoughtsTokenCount": 595,
"totalTokenCount": 1158 }, "invocationId": "e-7cfd3f16-
2522-4444-acc2-55afd9b5dc9f", "author": "imagine_agent",
"actions": { "stateDelta": {}, "artifactDelta": {},
"requestedAuthConfigs": {}, "requestedToolConfirmations":
{} }, "id": "a306c222-4e74-490d-a4de-55b31e41e95f",
"timestamp": 1759920695.189489 }


data: { "content": { "parts": [ { "text": "A hidden grotto
deep within an ancient forest, illuminated entirely by
clusters of various bioluminescent fungi casting a soft
emerald green and sapphire blue glow onto damp moss,
gnarled tree roots, and glistening droplets of water,
mystical atmosphere, hyperrealistic, cinematic lighting,
macro photography." } ], "role": "model" },
"usageMetadata": { "candidatesTokenCount": 56,
"promptTokenCount": 507, "promptTokensDetails": [ {
"modality": "TEXT", "tokenCount": 507 } ],
"thoughtsTokenCount": 595, "totalTokenCount": 1158 },
"invocationId": "e-7cfd3f16-2522-4444-acc2-55afd9b5dc9f",
"author": "imagine_agent", "actions": { "stateDelta": {},
"artifactDelta": {}, "requestedAuthConfigs": {},
"requestedToolConfirmations": {} }, "id": "d3b1d515-b996-
4c23-8dce-5dc60925f3b0", "timestamp": 1759920695.467105 }
```


การดีบักด้วย API Document

เซิร์ฟเวอร์ API จะสร้าง API Document แบบโต้ตอบโดยอัตโนมัติผ่าน *Swagger UI* ซึ่งเป็นเครื่องมือที่มีประโยชน์อย่างมากในการสำรวจ endpoint ต่างๆ ทำความเข้าใจรูปแบบของคำขอ (request

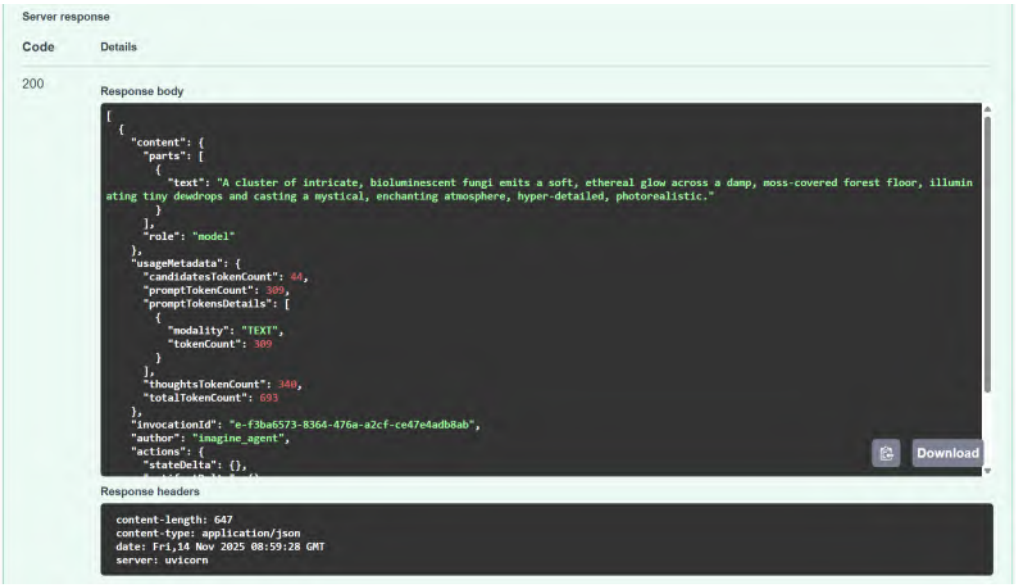
format) และทดสอบการทำงานของเอเจนต์ ได้โดยตรงจากเบราว์เซอร์ของคุณ

ในการใช้งานเอกสารแบบโต้ตอบ ให้เริ่มต้นเซิร์ฟเวอร์ API แล้วเปิดเว็บเบราว์เซอร์ไปที่ http://localhost:8000/docs



API Document

เมื่อเข้าไปแล้ว คุณจะเห็นรายการของ API endpoint ทั้งหมดที่มีอยู่ในระบบในรูปแบบแบบโต้ตอบ ซึ่งสามารถคลิกเพื่อดูรายละเอียดของพารามิเตอร์ (parameters), โครงสร้างของคำขอ (request body) และ รูปแบบการตอบกลับ (response schema) ได้อย่างครบถ้วน นอกจากนี้คุณยังสามารถคลิกที่ปุ่ม “Try it out” เพื่อส่งคำขอจริงไปยังเอเจนต์ที่กำลังทำงานอยู่ได้ทันที



ทดสอบโดยใช้ API Document

ในบทนี้ เราได้เรียนรู้พื้นฐานสำคัญของการสร้างเอเจนต์ด้วย Agent Development Kit (ADK) โดยเริ่มตั้งแต่การกำหนดตัวตนและวัตถุประสงค์ของเอเจนต์ผ่าน name, description และ model ที่จะใช้เป็นแกนหลักในการประมวลผล หัวใจสำคัญของการสร้างเอเจนต์คือการเขียน instruction ที่ชัดเจนและละเอียด เพื่อเป็นแนวทางให้เอเจนต์เข้าใจภารกิจหลัก ข้อจำกัด และวิธีการทำงาน

เราได้เห็นถึงพลังของเอเจนต์ในการเชื่อมต่อกับความสามารถภายนอกผ่านการเรียกใช้เครื่องมือ (Tools) ซึ่งมีทั้งเครื่องมือที่ ADK เตรียมไว้ให้ เช่น Google Search และการสร้างเครื่องมือขึ้นเองเพื่อเชื่อมต่อกับบริการภายนอกอย่าง Brave Search ผ่านไลบรารี LangChain นอกจากนี้ยังได้เรียนรู้วิธีการจัดการตัวแปรสถานะ (State Variable) เพื่อเพิ่มความยืดหยุ่นในการทำงาน

ท้ายที่สุด เราได้ลงมือสร้างเอเจนต์ตัวอย่าง ตั้งแต่ Imagine Agent ที่ใช้เพียง instruction ในการปรับปรุงพรอมต์ ไปจนถึง News Agent ที่สามารถค้นหาและสรุปข่าวสาร พร้อมทั้งสำรวจวิธีการรันเอเจนต์ในรูปแบบต่างๆ ไม่ว่าจะเป็น Web UI สำหรับการพัฒนา, Command Line สำหรับการโต้ตอบโดยตรง และ API Server สำหรับการนำไปเชื่อมต่อกับระบบอื่น ซึ่งทั้งหมดนี้เป็นพื้นฐานสำคัญในการพัฒนาเอเจนต์ที่มีความสามารถซับซ้อนต่อไป

การตั้งค่าและการควบคุมขั้นสูง

นอกเหนือจากพารามิเตอร์พื้นฐานแล้ว LlmAgent ยังมีตัวเลือกเพิ่มเติมสำหรับการควบคุมพฤติกรรมของเอเจนต์ในระดับที่ละเอียดขึ้น เพื่อให้สามารถปรับแต่งการทำงานให้เหมาะสมกับกรณีใช้งานที่ซับซ้อนยิ่งขึ้น

การปรับแต่งการสร้างข้อความของ LLM

สามารถควบคุมลักษณะการตอบสนองของโมเดลภาษาได้ผ่าน generate_content_config ซึ่งใช้สำหรับกำหนดพฤติกรรมการสร้างข้อความของ LLM โดยตรง

- **generate_content_config (Optional)** ใช้ส่งออบเจ็กต์ google.genai.types.GenerateContentConfig เพื่อกำหนดพารามิเตอร์ต่าง ๆ เช่น

- temperature ระดับความสุ่มของคำตอบ
- max_output_tokens ความยาวสูงสุดของผลลัพธ์
- top_p, top_k สำหรับการสุ่มเชิงความน่าจะเป็น
- safety_settings สำหรับควบคุมเนื้อหาที่อาจเป็นอันตราย

ตัวอย่างการตั้งค่าเพื่อให้ผลลัพธ์มีความคงที่และปลอดภัยมากขึ้นมีดังนี้

```
from google.genai import types

agent = LlmAgent(
    # ... other params
    generate_content_config=types.GenerateContentConfig(
        temperature=0.2, # More deterministic output
        max_output_tokens=250,
        safety_settings=[
            types.SafetySetting(

category=types.HarmCategory.HARM_CATEGORY_DANGEROUS_CONTEN
T,

threshold=types.HarmBlockThreshold.BLOCK_LOW_AND_ABOVE,
            )
        ]
    )
)
```

การจัดโครงสร้างข้อมูล

สำหรับกรณีที่ต้องการแลกเปลี่ยนข้อมูลกับ LLM ในรูปแบบที่มีโครงสร้างชัดเจน ADK รองรับการกำหนด schema ทั้งฝั่งอินพุตและเอาต์พุต เพื่อบังคับรูปแบบข้อมูลให้เป็นไปตามที่ต้องการ

- **input_schema (Optional)** ใช้กำหนดโครงสร้างของข้อมูลนำเข้า หากตั้งค่าไว้ ข้อความจากผู้ใช้ที่ส่งเข้าเอเจนต์จะต้องเป็น JSON ที่สอดคล้องกับ schema นี้ โดยคำสั่ง (instruction) ควรอธิบายรูปแบบข้อมูลให้ชัดเจน

- **output_schema (Optional)** ใช้กำหนดโครงสร้างของข้อมูลผลลัพธ์ โดยบังคับให้เอเจนต์ส่งคำตอบสุดท้ายออกมาเป็น JSON ตาม schema ที่กำหนด

- **output_key (Optional)** ใช้ระบุ key สำหรับบันทึกผลลัพธ์สุดท้ายของเอเจนต์ลงใน state ของ session โดยอัตโนมัติ เหมาะสำหรับการส่งต่อข้อมูลระหว่างเอเจนต์หรือขั้นตอนต่าง ๆ ใน workflow

ตัวอย่างการเข้าถึงค่าใน state

```
session.state[output_key] = agent_response_text
```

ตัวอย่างการกำหนด schema ด้วย Pydantic