

# ปั้น AI Agent ด้วย PydanticAI

เรียนรู้ครบจบในเล่มเดียว  
สู่การเป็นมืออาชีพ

อนุชิต ชโลธ

สำนักพิมพ์ท้อปวาท

# ปั้น AI Agent ด้วย PydanticAI

เรียนรู้ครบจบในเล่มเดียว สู่การเป็น  
มืออาชีพ

อนุชิต ชโลธร

(อัปเดตครั้งที่ 3)

# คำนำสำนักพิมพ์

ในนามของสำนักพิมพ์ เรามีความยินดีเป็นอย่างยิ่งที่ได้นำเสนอหนังสือ “ปั้น AI Agent ด้วย PydanticAI: เรียนรู้ครบจบในเล่มเดียว สู่การเป็นมืออาชีพ” สู่สายตานักพัฒนาและผู้สนใจในเทคโนโลยีปัญญาประดิษฐ์ทุกท่าน

ปฏิเสธไม่ได้ว่า Generative AI และ Large Language Models (LLMs) คือคลื่นลูกใหม่ที่กำลังเปลี่ยนแปลงโลกเทคโนโลยีในทุกมิติ ความสามารถในการสร้างแอปพลิเคชันที่สามารถ “คิด” และ “สื่อสาร” ได้ใกล้เคียงกับมนุษย์ได้เปิดประตูสู่ความเป็นไปได้ใหม่ๆ อย่างที่ไม่เคยมีมาก่อน อย่างไรก็ตาม การเปลี่ยนศักยภาพอันน่าทึ่งนี้ให้กลายเป็นผลิตภัณฑ์ที่ใช้งานได้จริงและเชื่อถือได้นั้น ยังคงเป็นความท้าทายที่ยิ่งใหญ่

PydanticAI ได้เข้ามามีบทบาทสำคัญในการลดช่องว่างดังกล่าว ด้วยการนำเสนอแนวทางที่สวยงามในการสร้างสะพานเชื่อมระหว่างโลกของ AI ที่ยืดหยุ่นและโลกของซอฟต์แวร์ที่ต้องการความแม่นยำและโครงสร้างที่ชัดเจน

หนังสือเล่มนี้ไม่ได้เป็นเพียงคู่มือการใช้งานไลบรารี แต่เป็นตำราอาหารที่รวบรวม “สูตรสำเร็จ” สำหรับการสร้าง AI Agent ในสถานการณ์ต่างๆ ที่นักพัฒนาต้องเผชิญในโลกแห่งความเป็นจริง ตั้งแต่การดึงข้อมูลพื้นฐาน, การสร้าง Chatbot, การทำงานกับฐานข้อมูลและ API, ไปจนถึงเทคนิคขั้นสูงสำหรับการทดสอบและนำขึ้นระบบโปรดักชัน ด้วยแนวทางการเขียนที่เน้นการลงมือทำและตัวอย่างที่จับต้องได้ เราเชื่อมั่นว่าหนังสือเล่มนี้จะเป็นเครื่องมือสำคัญที่ช่วยให้คุณสามารถนำ PydanticAI ไปประยุกต์ใช้ได้อย่างรวดเร็วและมั่นใจ

เราหวังว่าหนังสือเล่มนี้จะเป็นส่วนหนึ่งในการขับเคลื่อนวงการพัฒนาซอฟต์แวร์ AI ของไทยให้ก้าวไปข้างหน้า และช่วยให้คุณสามารถสร้างสรรค์นวัตกรรมที่ยอดเยียมออกมาได้ในที่สุด

ทีมงานสำนักพิมพ์

# คำนำ

ในยุคที่ Large Language Models (LLMs) ได้กลายเป็นเทคโนโลยีที่ก้าวล้ำและเข้าถึงได้ง่าย นักพัฒนาทั่วโลกต่างตื่นตัวกับศักยภาพในการสร้างสรรค์แอปพลิเคชันอัจฉริยะที่สามารถเข้าใจและโต้ตอบกับภาษามนุษย์ได้อย่างเป็นธรรมชาติ แต่ทว่า ท่ามกลางความมหัศจรรย์นี้ เรากลับพบความท้าทายสำคัญประการหนึ่ง นั่นคือ “ความไร้ระเบียบ” ของข้อมูลที่ได้จาก LLMs

LLMs สร้างข้อความที่อ่านสั่นไหวได้อย่างยอดเยี่ยม แต่ผลลัพธ์ที่ได้มักอยู่ในรูปแบบของข้อความดิบ (raw text) ที่ไม่มีโครงสร้างชัดเจน การจะนำข้อมูลเหล่านี้ไปใช้งานต่อในแอปพลิเคชันจริง เช่น การบันทึกลงฐานข้อมูล, การเรียกใช้ API หรือการแสดงผลบน UI จึงกลายเป็นเรื่องที่ซับซ้อนและเต็มไปด้วยโค้ดที่ต้องเขียนขึ้นเพื่อจัดการกับข้อมูลโดยเฉพาะ (boilerplate code)

PydanticAI แก้ปัญหาเรื่องเหล่านี้โดยเฉพาะ โดยใช้ปรัชญาอันเรียบง่ายของ Pydantic ซึ่งเป็นไลบรารีสำหรับการตรวจสอบและจัดการข้อมูลที่ดีที่สุดตัวหนึ่งของ Python เข้ากับ

ความสามารถของ LLMs ผลลัพธ์ที่ได้คือเครื่องมือที่ช่วยให้นักพัฒนาสามารถ “สั่ง” ให้ LLM คืนค่าผลลัพธ์ออกมาเป็น Pydantic Object ที่มีโครงสร้างชัดเจน, ตรวจสอบความถูกต้องได้ และพร้อมใช้งานต่อได้ทันที

หนังสือ “ปั้น AI Agent ด้วย PydanticAI: เรียนรู้ครบจบในเล่มเดียว สู่การเป็นมืออาชีพ” เล่มนี้ เป็นคู่มือที่จะพาคุณเดินทางตั้งแต่จุดเริ่มต้นของการทำความเข้าใจปัญหา ไปจนถึงการสร้าง AI Agent ที่ซับซ้อนและพร้อมสำหรับนำไปใช้งานจริง (Production-ready) เราจะเปลี่ยน “Prompt” ที่เป็นเพียงไอเดียในหัวของคุณ ให้กลายเป็น “Product” ที่จับต้องได้ ผ่านสูตรสำเร็จที่คัดสรรมาอย่างดี พร้อมตัวอย่างโค้ดที่นำไปใช้ได้จริง และคำอธิบายเบื้องหลังการทำงานที่ชัดเจน

ไม่ว่าคุณจะเป็นนักพัฒนาที่เพิ่งเริ่มต้นกับโลกของ AI หรือเป็นวิศวกรผู้มีประสบการณ์ที่กำลังมองหาเครื่องมือที่จะช่วยให้การทำงานกับ LLMs เป็นเรื่องง่ายและมีประสิทธิภาพมากขึ้น ผมหวังว่าหนังสือเล่มนี้จะเป็นเข็มทิศนำทางให้คุณสามารถปลดปล่อยศักยภาพของ AI และสร้างสรรค์ผลงานที่น่าทึ่งออกมาได้สำเร็จ

# สารบัญ

|                                 |           |
|---------------------------------|-----------|
| <b>แนะนำ PydanticAI</b>         | <b>1</b>  |
| จุดเด่นของ PydanticAI           | 2         |
| ก้าวแรกสู่ AI Agent             | 4         |
| <b>องค์ประกอบของ PydanticAI</b> | <b>11</b> |
| Agent                           | 11        |
| LLM Model                       | 12        |
| OpenAI                          | 12        |
| Google                          | 20        |
| Anthropic (Claude)              | 32        |
| Bedrock                         | 47        |
| Cerebras                        | 62        |
| Cohere                          | 64        |
| Groq                            | 66        |
| Hugging Face                    | 68        |
| Mistral                         | 69        |
| OpenRouter                      | 71        |

|                                                     |            |
|-----------------------------------------------------|------------|
| Outlines                                            | 74         |
| โมเดลที่ใช้งานร่วมกับ OpenAI API ได้                | 85         |
| การใช้งาน Docker Model Runner                       | 104        |
| Messages และ Chat History                           | 108        |
| การทำ ChatBot แบบไม่มีประวัติการสนทนา               | 108        |
| การทำ ChatBot แบบมีประวัติการสนทนา                  | 110        |
| การเพิ่ม System Prompt เพื่อกำหนด persona           | 111        |
| <b>การป้อนข้อมูลรูปภาพ เสียง วิดีโอ และเอกสาร</b>   | <b>114</b> |
| การป้อนข้อมูลรูปภาพ (Image Input)                   | 114        |
| การป้อนข้อมูลรูปภาพผ่าน URL (ImageUrl)              | 114        |
| การป้อนข้อมูลรูปภาพผ่านข้อมูลไบนารี (BinaryContent) | 115        |
| การป้อนข้อมูลเสียง (Audio Input)                    | 117        |
| การป้อนข้อมูลวิดีโอ (Video Input)                   | 117        |
| การป้อนข้อมูลเอกสาร (Document Input)                | 117        |
| การป้อนข้อมูลเอกสารผ่าน URL (DocumentUrl)           | 118        |
| การป้อนข้อมูลเอกสารผ่านไฟล์ไบนารี (BinaryContent)   | 120        |
| ดาวน์โหลดก่อนส่ง vs ส่ง URL โดยตรง                  | 123        |

|                                        |            |
|----------------------------------------|------------|
| <b>เอาต์พุต</b>                        | <b>125</b> |
| ตัวอย่างการกำหนด output_type เป็นโมเดล | 126        |
| ข้อมูลเอาต์พุต                         | 126        |
| ฟังก์ชันเอาต์พุต                       | 128        |
| ความแตกต่างกับ Tool Function ทั่วไป    | 129        |
| คุณสมบัติของฟังก์ชันเอาต์พุต           | 129        |
| เอาต์พุตเป็นข้อความ                    | 131        |
| โหมดการแสดงผล                          | 132        |
| การสตรีมผลลัพธ์                        | 137        |
| การสตรีมข้อความแบบปกติ                 | 138        |
| การสตรีมแบบ delta                      | 139        |
| การสตรีมข้อมูลโครงสร้าง                | 140        |
| Thinking (Reasoning)                   | 144        |
| OpenAI                                 | 145        |
| OpenAI Responses                       | 146        |
| Raw Reasoning ไม่มี Summary            | 147        |
| Anthropic                              | 148        |
| Google                                 | 149        |
| Bedrock                                | 149        |

|                                             |            |
|---------------------------------------------|------------|
| Groq                                        | 150        |
| OpenRouter                                  | 151        |
| Mistral                                     | 152        |
| Cohere                                      | 152        |
| Hugging Face                                | 152        |
| Outlines                                    | 153        |
| สรุป                                        | 153        |
| <b>เพิ่มศักยภาพให้ Agent ด้วยเครื่องมือ</b> | <b>155</b> |
| Function Tools คืออะไร                      | 155        |
| สร้างเครื่องมือคำนวณเลข                     | 157        |
| สร้างเครื่องมือดึงข้อมูลสภาพอากาศจาก API    | 160        |
| ผลลัพธ์จากเครื่องมือ                        | 163        |
| โครงสร้างของเครื่องมือ                      | 163        |
| การย่อ Schema เมื่อมีพารามิเตอร์เดียว       | 166        |
| ความสามารถขั้นสูงของเครื่องมือ              | 169        |
| ผลลัพธ์จากเครื่องมือ                        | 169        |
| การส่งคืนค่าขั้นสูงด้วย ToolReturn          | 172        |
| การกำหนด Schema ของเครื่องมือ               | 175        |
| เครื่องมือแบบไดนามิก                        | 177        |

|                                                               |     |
|---------------------------------------------------------------|-----|
| เครื่องมือแบบไดนามิกระดับ Agent                               | 178 |
| การทำงานของเครื่องมือและกลไกการลองใหม่                        | 179 |
| การตั้งเวลาในการทำงานของเครื่องมือ                            | 182 |
| การเรียกใช้เครื่องมือแบบขนานและการทำงาน<br>พร้อมกัน           | 184 |
| การจัดการฟังก์ชันแบบ Async และ Sync                           | 185 |
| การเรียกผลลัพธ์จากเครื่องเครื่องมือพร้อมกับ<br>เครื่องมืออื่น | 185 |
| การผูกเครื่องมือหลายตัวให้ Agent                              | 186 |
| การเรียกใช้ข้อมูลใน Context                                   | 189 |
| Agent เลือกใช้เครื่องมืออย่างไร                               | 193 |
| ชุดเครื่องมือ                                                 | 194 |
| วิธีการกำหนดชุดเครื่องมือให้กับ Agent                         | 195 |
| ฟังก์ชันชุดเครื่องมือ                                         | 198 |
| การประกอบชุดเครื่องมือ                                        | 201 |
| การรวมชุดเครื่องมือ                                           | 201 |
| การกรองเครื่องมือ                                             | 202 |
| การเพิ่ม Prefix ให้ชื่อเครื่องมือ                             | 203 |
| การเปลี่ยนชื่อเครื่องมือ                                      | 205 |

|                                                                    |     |
|--------------------------------------------------------------------|-----|
| การกำหนดเครื่องมือแบบไดนามิก                                       | 207 |
| การกำหนดให้ต้องขออนุมัติก่อนเรียกใช้เครื่องมือ                     | 211 |
| การเปลี่ยนพฤติกรรมในการเรียกใช้เครื่องมือ                          | 214 |
| ชุดเครื่องมือภายนอก                                                | 217 |
| การใช้งานเครื่องมือภายนอกและการใช้เครื่องมือ<br>ภายหลังร่วมกับ API | 220 |
| การสร้างชุดเครื่องมือแบบไดนามิก                                    | 225 |
| การเรียกใช้เครื่องมือในภายหลัง                                     | 228 |
| Human-in-the-Loop: การขออนุมัติใช้เครื่องมือ                       | 230 |
| ตัวอย่างขออนุมัติการลบไฟล์ทั้งหมด และการแก้ไข<br>ไฟล์ที่ถูกป้องกัน | 232 |
| External Tool Execution                                            | 240 |
| ตัวอย่างส่งงานที่ใช้เวลานานไปประมวลผลเบื้อง<br>หลัง                | 242 |
| เครื่องมือจากผู้ให้บริการโมเดล                                     | 248 |
| การรองรับจากผู้ให้บริการ (Provider Support)                        | 249 |
| การตั้งค่าแบบไดนามิก (Dynamic Configuration)                       | 250 |
| Web Search Tool                                                    | 252 |
| Code Execution Tool                                                | 257 |

|                                       |            |
|---------------------------------------|------------|
| Image Generation Tool                 | 262        |
| Web Fetch Tool                        | 269        |
| Memory Tool                           | 273        |
| MCP Server Tool                       | 277        |
| File Search Tool                      | 282        |
| เครื่องมือที่มีมาให้ในตัว             | 286        |
| DuckDuckGo Search Tool                | 287        |
| Tavily Search Tool                    | 289        |
| สรุปเครื่องมือที่มีมาให้ในตัว         | 291        |
| เครื่องมือจากผู้ให้บริการอื่นๆ        | 291        |
| MCP (Model Context Protocol) Tools    | 292        |
| เครื่องมือจาก LangChain               | 319        |
| เครื่องมือจาก ACL.dev                 | 321        |
| การใช้เครื่องมือใน Docker MCP Gateway | 324        |
| <b>การแปลงข้อความให้เป็นเวกเตอร์</b>  | <b>329</b> |
| การใช้งาน                             | 329        |
| Queries vs Documents                  | 331        |
| ผลลัพธ์การค้นหา                       | 331        |
| ผู้ให้บริการโมเดล Embedding           | 333        |

|                                           |            |
|-------------------------------------------|------------|
| OpenAI                                    | 333        |
| Cohere                                    | 337        |
| การใช้โมเดล Sentence Transformers         | 339        |
| การติดตั้งไลบรารี                         | 339        |
| การตั้งค่าโมเดล                           | 339        |
| การเลือกอุปกรณ์ CPU/GPU                   | 340        |
| การเรียกใช้โมเดลของตนเอง                  | 341        |
| การตั้งค่าเพิ่มเติมผ่าน EmbeddingSettings | 341        |
| การนับจำนวนโทเคน                          | 342        |
| การทดสอบ Embedding                        | 343        |
| การติดตามการทำงาน                         | 344        |
| <b>การสร้าง Chatbot</b>                   | <b>346</b> |
| สร้าง Agent ที่จดจำบริบทและโต้ตอบได้      | 346        |
| ตัวอย่างสร้างรูปสนทนา                     | 347        |
| ทดลองคุยกับ Chatbot                       | 349        |
| สร้าง Chatbot ที่มีบุคลิก (Persona)       | 350        |
| ใช้ System Prompt เพื่อกำหนดบุคลิก        | 350        |
| สร้าง Agent บุคลิกกับปัดนแจ๊ค             | 350        |
| ทดลองคุยกับกับปัดนแจ๊ค                    | 351        |

|                                                       |            |
|-------------------------------------------------------|------------|
| <b>สร้างระบบ RAG (Retrieval-Augmented Generation)</b> | <b>354</b> |
| RAG คืออะไร                                           | 354        |
| ข้อดีของ RAG                                          | 356        |
| การใช้งาน RAG Agent ร่วมกับ FAISS                     | 356        |
| สร้างคลังความรู้จากเอกสาร                             | 357        |
| เตรียมเครื่องมือและติดตั้งไลบรารี                     | 358        |
| ขั้นตอนสร้าง Vector Store                             | 358        |
| สร้าง Agent ที่ค้นข้อมูลจากเอกสารเพื่อตอบคำถาม        | 361        |
| การใช้งาน RAG Agent ร่วมกับ Qdrant                    | 364        |
| ทำไมต้องใช้ Qdrant แทน FAISS?                         | 365        |
| วิธีการใช้งาน Qdrant ร่วมกับ RAG Agent                | 366        |
| สร้างคลังความรู้จากเอกสาร                             | 367        |
| สร้าง Agent ที่ค้นข้อมูลจากเอกสารเพื่อตอบคำถาม        | 372        |
| เทคนิคการแปลงไฟล์เอกสารเป็น Markdown                  | 375        |
| การใช้งานผ่าน Python API                              | 375        |
| การใช้งานผ่าน Command Line Interface (CLI)            | 377        |
| <b>การส่งงานฐานข้อมูล (SQL Generation)</b>            | <b>379</b> |
| สร้าง Tool สำหรับเชื่อมต่อกับฐานข้อมูล SQL            | 379        |

|                                           |            |
|-------------------------------------------|------------|
| เตรียมฐานข้อมูลตัวอย่าง                   | 380        |
| สร้างฟังก์ชันสำหรับรัน SQL Query          | 383        |
| สร้าง Agent ที่แปลงคำถามให้เป็น SQL Query | 384        |
| <b>การสร้าง API ด้วย FastAPI</b>          | <b>390</b> |
| เตรียมความพร้อม                           | 390        |
| นำ Agent ไปใช้ใน FastAPI Endpoint         | 393        |
| สร้าง API แบบ Streaming                   | 396        |
| <b>การทำงานร่วมกันของ Agent หลายตัว</b>   | <b>400</b> |
| ทำไมต้องใช้ Agent หลายตัว                 | 400        |
| สร้าง Workflow Agents                     | 402        |
| <b>กราฟ (Graphs)</b>                      | <b>408</b> |
| การติดตั้ง                                | 408        |
| องค์ประกอบหลัก                            | 409        |
| GraphRunContext                           | 409        |
| End                                       | 409        |
| Node                                      | 409        |
| Graph                                     | 412        |
| การสร้าง Mermaid Diagram                  | 413        |
| กราฟที่มีสถานะ (Stateful Graphs)          | 414        |

|                                                      |     |
|------------------------------------------------------|-----|
| ตัวอย่าง เครื่องขายสินค้าอัตโนมัติ (Vending Machine) | 415 |
| การใช้งานกราฟร่วมกับ PydanticAI                      | 420 |
| ตัวอย่างการใช้งานกราฟ                                | 421 |
| เรียกใช้ไลบรารี pydantic_graph                       | 421 |
| นิยามข้อมูลผู้ใช้งานและอีเมล                         | 422 |
| กำหนดสถานะภายใน Graph                                | 422 |
| สร้าง Agent สำหรับเขียนอีเมล                         | 423 |
| สร้าง Node: WriteEmail                               | 423 |
| สร้าง Agent สำหรับให้ feedback                       | 424 |
| สร้าง Node: Feedback                                 | 424 |
| การเรียกใช้งานกราฟ                                   | 425 |
| การวนลูปทำงานโดยใช้กราฟ (Graph Iteration)            | 425 |
| ตัวอย่างกราฟนับถอยหลัง                               | 425 |
| การเรียกใช้งาน                                       | 426 |
| การควบคุมการรัน node ด้วย GraphRun.next()            | 427 |
| การบันทึกสถานะเพื่อ Resume                           | 427 |
| ตัวเลือกการบันทึกสถานะ                               | 428 |
| ตัวอย่างการ Resume จาก JSON File                     | 428 |

|                                                               |            |
|---------------------------------------------------------------|------------|
| <b>การทดสอบและประเมินผล</b>                                   | <b>431</b> |
| ทำไมการประเมินผลจึงสำคัญกับ LLM Application                   | 432        |
| การสร้างชุดข้อมูล (Dataset) สำหรับทดสอบด้วย<br>pydantic_evals | 433        |
| การรัน Evaluator เพื่อวัดความแม่นยำของ Agent                  | 435        |
| <b>การ Debug และ Monitoring</b>                               | <b>439</b> |
| การใช้ Logfire เพื่อติดตามการทำงานของ Agent                   | 440        |
| เทคนิคการเขียน Unit Test สำหรับ Tools และ Agent               | 444        |
| การทดสอบ Tools                                                | 444        |
| การทดสอบ Agent                                                | 445        |
| การใช้งาน OpenTelemetry กับ Pydantic AI                       | 446        |
| การใช้งาน Logfire ร่วมกับ otel-tui                            | 447        |
| การใช้งาน Logfire ร่วมกับ Opik                                | 449        |
| <b>โปรโตคอล Agent2Agent (A2A)</b>                             | <b>452</b> |
| FastA2A                                                       | 455        |
| การออกแบบ                                                     | 455        |
| ส่วนประกอบ A2A Protocol                                       | 456        |
| องค์ประกอบของ FastA2A                                         | 456        |
| ลักษณะการทำงานของ context_id                                  | 458        |

|                                                |            |
|------------------------------------------------|------------|
| สถาปัตยกรรมของ Storage                         | 459        |
| การใช้งานร่วมกับ PydanticAI                    | 459        |
| ตัวอย่างโค้ด Pirate Agent                      | 460        |
| ตัวอย่างโค้ด A2A Sever                         | 460        |
| ทดสอบใช้งาน                                    | 464        |
| <b>โปรโตคอล Agent User Interaction (AG-UI)</b> | <b>466</b> |
| ทำไมต้องใช้ AG-UI                              | 466        |
| การเชื่อมต่อที่มีอยู่แล้ว                      | 467        |
| การสร้างแอปด้วย create-ag-ui-app               | 470        |
| การสร้างแอป CLI                                | 471        |
| การสร้างเว็บแอปด้วย CopilotKit และ Next.js     | 476        |
| <b>การใช้งาน PydanticAI CLI</b>                | <b>485</b> |
| การติดตั้งและการตั้งค่าเบื้องต้น               | 485        |
| ตัวอย่างการเริ่มต้นและสนทนาเบื้องต้น           | 486        |
| คำสั่งพิเศษในโหมดโต้ตอบ                        | 487        |
| ตัวอย่างการใช้งาน                              | 489        |
| การระบุโมเดลและเรียกใช้งาน Agent               | 489        |
| การใช้งานร่วมกับ Agent                         | 490        |
| การเรียกใช้ CLI โดยตรงจาก Agent Instance       | 492        |

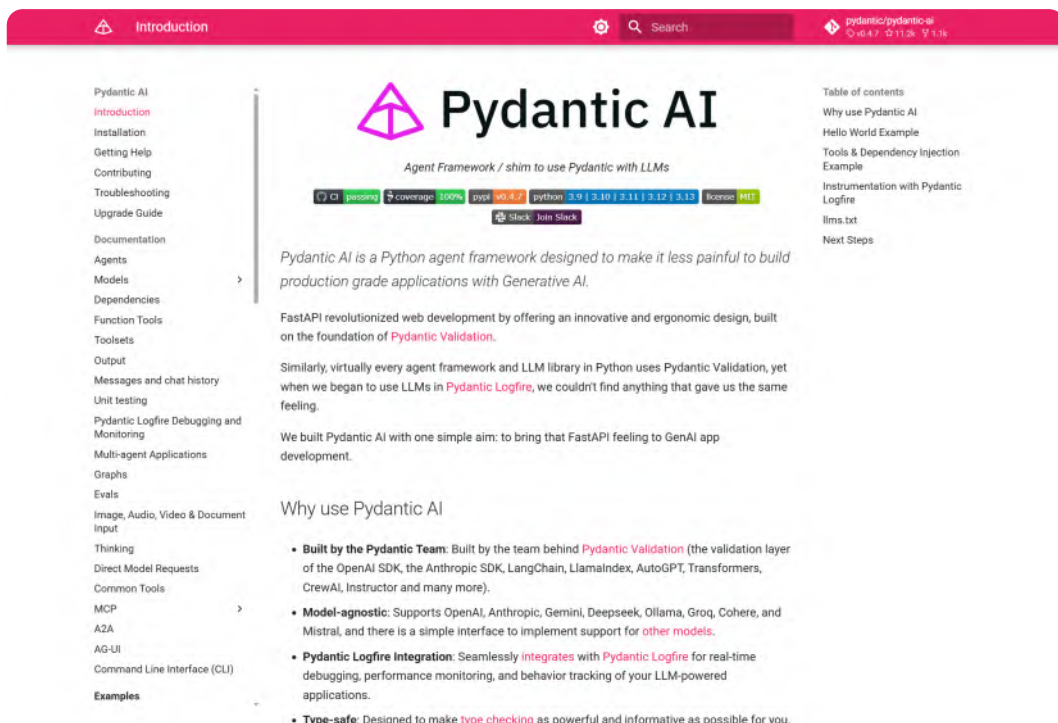
|                                                    |            |
|----------------------------------------------------|------------|
| Web Chat UI                                        | 493        |
| การติดตั้ง (Installation)                          | 493        |
| การใช้งานพื้นฐาน (Basic Usage)                     | 494        |
| ตัวอย่างการใช้งาน                                  | 494        |
| การตั้งค่าโมเดล (Configuring Models)               | 495        |
| ตัวอย่างการตั้งค่าโมเดล                            | 495        |
| การรองรับ Built-in Tools                           | 496        |
| ตัวอย่างรองรับ Built-in Tools                      | 496        |
| Memory Tool                                        | 497        |
| คำสั่งเพิ่มเติม (Extra Instructions)               | 497        |
| เส้นทางที่สงวนไว้ (Reserved Routes)                | 498        |
| การเรียกใช้ WebUI ผ่าน Clai                        | 499        |
| การให้บริการ Agent ที่มีอยู่แล้ว                   | 499        |
| ตัวอย่างรูปแบบการเรียกใช้งาน                       | 500        |
| Memory Tool                                        | 500        |
| ตัวเลือกของ Web UI (Web UI Options)                | 501        |
| การเรียกใช้งานแบบ Programmatic                     | 502        |
| ดูตัวเลือกทั้งหมด                                  | 502        |
| <b>สร้างแอปแชทบอตด้วย Streamlit และ PydanticAI</b> | <b>504</b> |

|                                                       |            |
|-------------------------------------------------------|------------|
| ติดตั้ง Library                                       | 505        |
| ตั้งค่า OpenAI API Key                                | 505        |
| สร้างไฟล์แอป                                          | 505        |
| อธิบายโค้ด                                            | 512        |
| การติดตั้งและตั้งค่าเบื้องต้น                         | 512        |
| การตั้งค่า UI เบื้องต้นของหน้าเว็บ                    | 513        |
| ตรวจสอบและขอ API Key จากผู้ใช้ (หากยังไม่มี)          | 513        |
| สร้าง Agent ของ PydanticAI                            | 514        |
| เตรียม session state สำหรับเก็บข้อความและประวัติสนทนา | 515        |
| แสดงข้อความเก่า                                       | 515        |
| ฟังก์ชันจำลองการพิมพ์แบบไหล (streaming)               | 515        |
| ฟังก์ชันเรียกใช้งาน Agent แบบ async                   | 516        |
| รับอินพุตจากผู้ใช้ และโต้ตอบกับ Agent                 | 516        |
| ส่วน Sidebar สำหรับควบคุมและข้อมูลสถานะ               | 517        |
| ปุ่ม Refresh หน้าแชท                                  | 518        |
| การใช้งาน                                             | 518        |
| <b>การ Deploy PydanticAI App</b>                      | <b>521</b> |
| การ Deploy API ด้วย Docker Compose                    | 521        |

|                                                 |            |
|-------------------------------------------------|------------|
| ตัวอย่างโค้ด AI Agent API                       | 521        |
| สร้าง Dockerfile สำหรับ Build Container         | 523        |
| สร้าง docker-compose.yaml สำหรับจัดการ Service  | 524        |
| สั่ง Build Image และ Run Container              | 525        |
| การ Deploy Web App                              | 527        |
| ตัวอย่างโค้ด Web App ด้วย Streamlit             | 527        |
| สร้าง Dockerfile สำหรับ Build Container         | 534        |
| สร้าง docker-compose.yaml สำหรับจัดการ Service  | 535        |
| สั่ง Build Image และ Run Container              | 535        |
| การ Deploy แบบ Agent CLI                        | 536        |
| ตัวอย่างโค้ด Agent CLI                          | 536        |
| สร้าง Dockerfile                                | 537        |
| สร้าง Docker Image                              | 538        |
| สั่ง Run Agent CLI โดยใช้ Docker                | 538        |
| <b>บทส่งท้าย การเดินทางของคุณกับ PydanticAI</b> | <b>540</b> |
| <b>ดาวนโหลดซอร์สโค้ด</b>                        | <b>542</b> |

# แนะนำ PydanticAI

ถ้า FastAPI คือเฟรมเวิร์กที่ทำให้การสร้าง API เป็นเรื่องสนุก PydanticAI ก็เป็นเครื่องมือสร้าง AI Agent ได้สนุกในแบบเดียวกัน



เว็บไซต์ [ai.pydantic.dev](https://ai.pydantic.dev)

PydanticAI เป็นเฟรมเวิร์กสำหรับสร้าง AI Agent ด้วยภาษา Python ออกแบบมาเพื่อช่วยให้นักพัฒนาสามารถสร้างแอปพลิเคชัน Generative AI ในระดับ Production ได้อย่างรวดเร็ว ถูกต้อง และมีโครงสร้างที่ดี โดยอาศัยจุดแข็งของ

Pydantic ซึ่งเป็นเครื่องมือที่คุ้นเคยกันดีในวงการภาษา Python และถูกใช้อย่างแพร่หลายในไลบรารี AI ชั้นนำ

แม้ว่าทุกวันนี้ไลบรารี LLM ต่างก็ใช้ Pydantic ในการจัดการข้อมูลอยู่เบื้องหลัง แต่กลับยังไม่มีเฟรมเวิร์กตัวไหนที่มอบประสบการณ์การพัฒนาในแบบที่ FastAPI เคยทำได้ นั่นจึงเป็นที่มาของ **PydanticAI**

## จุดเด่นของ PydanticAI

- **PydanticAI สร้างโดยทีม Pydantic โดยตรง**  
PydanticAI พัฒนาขึ้นโดยทีมเดียวกับผู้สร้าง Pydantic ไลบรารีตรวจสอบข้อมูลที่กลายเป็นหัวใจในไลบรารี AI ชั้นนำหลากหลายตัว เช่น OpenAI SDK, LangChain, LlamaIndex, Instructor, CrewAI และอื่นๆ อีกมากมาย
- **รองรับหลายโมเดล** PydanticAI สามารถเชื่อมต่อกับโมเดลยอดนิยมได้หลากหลาย เช่น OpenAI, Anthropic, Gemini, Ollama, Deepseek, Groq, Cohere และ Mistral พร้อมระบบอินเตอร์เฟซที่ยืดหยุ่น สามารถขยายให้รองรับโมเดลอื่นๆ ได้ง่าย

- **ทำงานร่วมกับ Pydantic Logfire ได้อย่างไร้รอยต่อ**  
สามารถเชื่อมต่อกับ Pydantic Logfire เพื่อดูการทำงานของ Agent แบบเรียลไทม์ ตรวจสอบประสิทธิภาพ และ พฤติกรรมของ LLM ได้สะดวกสบาย
- **ปลอดภัยด้วยระบบ Type Checking Agent และ**  
Output ในระบบของ PydanticAI ถูกออกแบบให้รองรับ Type-safe อย่างเต็มรูปแบบ ทำให้โค้ดของคุณสามารถ ตรวจสอบได้ตั้งแต่ก่อนรันจริง ช่วยลดข้อผิดพลาดในระบบ
- **เขียนได้ง่ายในสไตล์ Python** เฟรมเวิร์กนี้ออกแบบโดยยึด แนวคิด Pythonic ไม่ว่าจะเป็น Control Flow การจัดการ Agent แบบ Composition หรือแม้แต่ Dependency Injection ก็สามารถเขียนในรูปแบบที่คุณคุ้นเคยได้
- **จัดการผลลัพธ์ด้วยโครงสร้างที่แน่นอน** การใช้ Pydantic มาช่วยในการตรวจสอบและจัดรูปแบบผลลัพธ์จากโมเดล LLM ทำให้คุณมั่นใจได้ว่า Response ที่ได้จากโมเดลจะมีความสม่ำเสมอ และสามารถนำไปใช้งานต่อได้ทันที
- **ระบบ Dependency Injection** สามารถส่งข้อมูลหรือ บริการต่างๆ ไปยัง Agent ได้ผ่านระบบ Dependency Injection ซึ่งเหมาะอย่างยิ่งสำหรับการทำ Testing, Mocking และการพัฒนาแบบ Iterative

- **รองรับการสตรีมผลลัพธ์จากโมเดล** คุณสามารถรับผลลัพธ์จาก LLM แบบสตรีม (Streamed Response) ได้ทันที พร้อมระบบตรวจสอบข้อมูลในระหว่างการส่ง ทำให้เหมาะสำหรับแอปที่ต้องตอบสนองแบบเรียลไทม์
- **เขียนโค้ดแบบกราฟด้วย Pydantic Graph** สำหรับแอปที่มีความซับซ้อน PydanticAI ยังรองรับการสร้างโฟลว์แบบกราฟด้วย Type Hint ช่วยให้การจัดการลำดับขั้นตอนต่างๆ มีความชัดเจน ลดปัญหาโค้ดพันกัน (Spaghetti Code)

PydanticAI คือเฟรมเวิร์กที่ออกแบบมาสำหรับนักพัฒนา Python ที่ต้องการพัฒนา AI Agent อย่างจริงจัง ให้ประสบการณ์แบบเดียวกับการใช้ FastAPI แต่เป็นโลกของ Generative AI

หากคุณกำลังมองหาเครื่องมือที่ช่วยให้การทำงานกับ LLM เป็นระบบมากขึ้น ใช้ซ้ำได้ ตรวจสอบได้ และขยายได้ในระยะยาว *PydanticAI* คือคำตอบที่คุณไม่ควรมองข้าม

## ก้าวแรกสู่ AI Agent

ถ้าในโลกของการเขียนโปรแกรมมี “Hello, World!” เป็นบทเรียนเบื้องต้น การสร้าง AI Agent ก็เริ่มต้นจากสิ่งที่สำคัญไม่แพ้กัน นั่นคือ การสกัดข้อมูลแบบมีโครงสร้าง (Structured Data Extraction)

หนึ่งในความท้าทายในการทำงานร่วมกับ Large Language Models (LLMs) คือการรับมือกับผลลัพธ์ที่อยู่ในรูปแบบของ “ข้อความอิสระ” (Unstructured Text) ซึ่งยากต่อการนำไปใช้งานในระบบซอฟต์แวร์แบบดั้งเดิม นี่คือจุดที่ PydanticAI เข้ามามีบทบาทสำคัญ โดยทำหน้าที่เป็นสะพานเชื่อมระหว่างโลกของข้อความที่คลุมเครือของ LLM กับโลกของข้อมูลแบบมีโครงสร้างชัดเจนในแบบของ Pydantic

เราจะเริ่มต้นจากตัวอย่างง่ายๆ เพื่อเรียนรู้การใช้งาน PydanticAI อย่างเป็นรูปธรรม โดยการสร้าง Agent ที่สามารถอ่านข้อความธรรมดา แล้วสกัดข้อมูลชื่อ อายุ และแปลงข้อมูลที่ได้ ให้อยู่ในรูปแบบของ Pydantic Object ที่สามารถใช้งานและตรวจสอบความถูกต้องได้ทันที

เครื่องมือที่ต้องใช้มีดังนี้

- Python 3.9 ขึ้นไป

- ไลบรารี pydantic-ai และ openai
- OpenAI API Key

เริ่มจากการติดตั้งไลบรารีที่จำเป็น ดังนี้

```
pip install "pydantic-ai-slim[openai]"
```

จากนั้นให้ตั้งค่า OpenAI API Key ของคุณเป็น environment variable ดังนี้

สำหรับ Linux หรือ macOS

```
export OPENAI_API_KEY="sk-..."
```

สำหรับ Windows (Command Prompt)

```
set OPENAI_API_KEY="sk-..."
```

สำหรับ Windows (Powershell)

```
$env:OPENAI_API_KEY="sk-..."
```

นิยามโครงสร้างข้อมูลด้วย Pydantic โดยการออกแบบ “พิมพ์เขียว” ให้กับข้อมูลที่เราต้องการให้ LLM ช่วยดึงออกมา ดังนี้

สร้างไฟล์ชื่อ extract\_user.py แล้วใส่โค้ดโมเดล User ต่อไปนี้

```
from pydantic import BaseModel, Field

class User(BaseModel):
    """
    A model for storing user data extracted from text
    The text in this docstring will be passed to LLM
    for understanding
    """
    name: str = Field(description="Full name and surname of the user")
    age: int = Field(description="Age of the user in years")
```

การระบุ docstring และ description ชัดเจนในแต่ละฟิลด์ช่วยให้ LLM เข้าใจว่าแต่ละฟิลด์หมายถึงอะไร และควรดึงข้อมูลแบบไหน

จากนั้นเราจะสร้าง Agent ที่ทำหน้าที่ดึงข้อมูลจากข้อความให้เป็นไปตามโมเดล User ที่นิยามไว้ เพิ่มโค้ดลงในไฟล์ `extract_user.py` ดังนี้

```
from pydantic_ai import Agent
from pydantic_ai.models.openai import OpenAIModel

llm = OpenAIModel('gpt-4o')
agent = Agent(llm)

prompt = "My name is John Doe and I will be 30 years
old on my next birthday."

print(f"Input Prompt: '{prompt}'\n")

agent_run_result = agent.run_sync(prompt,
output_type=User)
result: User = agent_run_result.output

print("--- Extracted Data ---")
print(f"Name: {result.name}")
print(f"Age: {result.age}")
print(f"\nType of result: {type(result)}")
print("Object: ", repr(result))
print("-----")
```

รันทดสอบจาก Terminal รันไฟล์ด้วยคำสั่ง

```
python extract_user.py
```

คุณ将会เห็นผลลัพธ์ในลักษณะนี้

```
Input Prompt: 'My name is John Doe and I will be 30
years old on my next birthday.'
```

```
--- Extracted Data ---
```

```
Name: John Doe
```

```
Age: 29
```

```
Type of result: <class '__main__.User'>
```

```
Object: User(name='John Doe', age=29)
```

```
-----
```

วิเคราะห์ผลลัพธ์

1. **ความถูกต้องของข้อมูล** Agent สามารถสกัดชื่อ John Doe และอายุ 29 ได้อย่างแม่นยำ ในข้อความจะใช้คำว่า “will be 30” ซึ่งหมายถึงในอนาคตจะอายุ 30
2. **ผลลัพธ์เป็นวัตถุ Pydantic** ตัวแปร result ไม่ใช่แค่ string หรือ dict แต่เป็นอินสแตนซ์ของคลาส User ทำให้สามารถ

เรียกใช้ `result.name` และ `result.age` ได้โดยตรง พร้อมระบบตรวจสอบข้อมูลและ Type Hint ของ Pydantic

เบื้องหลัง PydanticAI ทำงานดังนี้

- แปลงโมเดล User ให้เป็น Schema ที่ LLM เข้าใจ เหมือน Function Calling
- สร้าง Prompt ที่ประกอบด้วย Schema และคำสั่งเพื่อให้ LLM ตอบกลับข้อมูลในโครงสร้างที่กำหนด
- รับค่าตอบกลับ มักเป็น JSON string และนำมาสร้างเป็นอินสแตนซ์ของ User พร้อมตรวจสอบความถูกต้องให้อัตโนมัติ

PydanticAI ช่วยให้เราสามารถ “ประกาศว่าเราต้องการข้อมูลแบบไหน” และปล่อยให้ Agent จัดการส่วนที่ยุ่งยากในการสื่อสารกับ LLM แทนเรา

ในบทถัดไป เราจะเจาะลึกองค์ประกอบต่างๆ ของ Agent รวมถึงการเชื่อมต่อกับ LLM จากผู้ให้บริการอื่น เพื่อขยายความสามารถของระบบต่อไป

# องค์ประกอบของ PydanticAI

เรามาเจาะลึกองค์ประกอบสำคัญของ PydanticAI ซึ่งเป็นรากฐานในการสร้างแอปพลิเคชันที่ขับเคลื่อนด้วยปัญญาประดิษฐ์ ความเข้าใจในองค์ประกอบเหล่านี้จะช่วยให้คุณสามารถนำ PydanticAI ไปปรับใช้ในงานต่างๆ ได้อย่างยืดหยุ่น มีประสิทธิภาพ และสามารถขยายต่อขยายได้ในอนาคต

PydanticAI มีองค์ประกอบหลัก ดังนี้

1. **Agent** ตัวควบคุมหลัก หรือ “สมอง” ของระบบ
2. **LLM Models** ส่วนที่เชื่อมต่อกับผู้ให้บริการ Large Language Models
3. **Messages และ Chat History** กลไกการจัดการบริบทของบทสนทนา

## Agent

Agent คือคลาสหลักที่ทำหน้าที่เป็นศูนย์กลางการประมวลผลทั้งหมดใน PydanticAI ทำหน้าที่รับคำสั่งจากผู้ใช้ (prompt), ทำงานประสานกับ LLM, ตัดสินใจว่าจะเรียกใช้เครื่องมือ

(Tools) ใดบ้าง และจัดการส่งผลลัพธ์กลับไปยังผู้ใช้ อย่างมีแบบแผน

กระบวนการทำงานของ Agent ประกอบด้วย

- รับข้อมูลจากผู้ใช้
- สร้าง prompt ที่เหมาะสมสำหรับส่งไปยัง LLM
- เรียกใช้งาน LLM เพื่อประมวลผลคำตอบ
- ตรวจสอบผลลัพธ์ที่ได้ และตัดสินใจว่าจะตอบกลับหรือเรียกใช้เครื่องมือเพิ่มเติม
- ส่งผลลัพธ์สุดท้ายกลับในรูปแบบข้อมูลที่โครงสร้างชัดเจนด้วย Pydantic Model

## LLM Model

PydanticAI ถูกออกแบบให้สามารถทำงานร่วมกับ LLM จากหลากหลายผู้ให้บริการได้อย่างยืดหยุ่น ผ่านคลาส Model ซึ่งเป็นตัวกลางในการสื่อสารกับ API ของแต่ละราย เช่น OpenAI, Google (Gemini) หรือ Anthropic (Claude) เป็นต้น ทำให้สามารถสลับเปลี่ยนโมเดลได้โดยไม่ต้องแก้ไขโค้ดมาก

## OpenAI

ติดตั้ง OpenAI Model ดังนี้

```
pip install "pydantic-ai-slim[openai]"
```

ตั้งค่า API Key ของ OpenAI ใน environment variable  
ดังนี้

```
export OPENAI_API_KEY='your-api-key'
```

เชื่อมต่อกับโมเดลของ OpenAI อย่างเช่น gpt-5 สามารถ  
ทำได้ดังนี้

```
from pydantic_ai import Agent  
  
agent = Agent('openai:gpt-5')
```

หรือเรียกใช้ผ่าน OpenAIModel ดังนี้

```
from pydantic_ai import Agent  
from pydantic_ai.models.openai import OpenAIModel
```

```
model = OpenAIModel('gpt-5')
agent = Agent(model)
```

คุณสามารถกำหนด API Key ผ่าน provider ดังนี้

```
from pydantic_ai import Agent
from pydantic_ai.models.openai import OpenAIModel
from pydantic_ai.providers.openai import OpenAIProvider

# แนะนำให้ตั้งค่า API Key ผ่าน environment variable
provider = OpenAIProvider(api_key='your-api-key')
model = OpenAIModel('gpt-5', provider=provider)
agent = Agent(model)
```

## OpenAI Responses API

Pydantic AI รองรับการใช้งาน **OpenAI Responses API** โดยตรง ซึ่งเป็น API รุ่นใหม่ของ OpenAI ที่ออกแบบมาให้อจัดการบริบทการสนทนา เครื่องมือ (tools) และเหตุผลการตอบ (reasoning) ได้อย่างมีประสิทธิภาพมากขึ้น

คุณสามารถใช้งาน OpenAIResponsesModel ได้ทั้งในรูปแบบการอ้างอิงชื่อโมเดลโดยตรง หรือการสร้างอินสแตนซ์ของ

## โมเดลด้วยตนเอง

### การใช้งานด้วยชื่อโมเดลโดยตรง

```
from pydantic_ai import Agent
from pydantic_ai.models.openai import OpenAIResponsesModel

model = OpenAIResponsesModel('gpt-5')
agent = Agent(model)
...
```

## Built-in Tools ของ Responses API

Responses API มาพร้อม **Built-in Tools** ที่สามารถใช้งานได้ทันที โดยไม่จำเป็นต้องพัฒนาเครื่องมือเอง ได้แก่

- **Web search** — ให้โมเดลค้นหาข้อมูลล่าสุดจากเว็บก่อนสร้างคำตอบ
- **Code interpreter** — ให้โมเดลเขียนและรันโค้ด Python ใน sandbox ก่อนตอบกลับ
- **Image generation** — ให้โมเดลสร้างภาพจากข้อความ
- **File search** — ให้โมเดลค้นหาข้อมูลจากไฟล์ของคุณ

- **Computer use** — ให้โมเดลควบคุมคอมพิวเตอร์เพื่อทำงานแทนผู้ใช้

เครื่องมือ Web search, Code interpreter, Image generation และ File search รองรับแบบ native ผ่านฟีเจอร์

**Built-in tools** ของ Pydantic AI

## การเปิดใช้งาน Computer Use

เครื่องมือ **Computer use** สามารถเปิดใช้งานได้โดยกำหนด `ComputerToolParam` ใน `openai_builtin_tools` ของ `OpenAIResponsesModelSettings`

**หมายเหตุ** ในปัจจุบัน `Computer use` ยังไม่สร้าง `BuiltinToolCallPart` หรือ `BuiltinToolReturnPart` ใน `message history` และไม่รองรับ `streamed events`

```
from openai.types.responses import ComputerToolParam

from pydantic_ai import Agent
from pydantic_ai.models.openai import OpenAIResponsesModel, OpenAIResponsesModelSettings

model_settings = OpenAIResponsesModelSettings(
```

```

openai_builtin_tools=[
    ComputerToolParam(
        type='computer_use',
    )
],
)

model = OpenAIResponsesModel('gpt-5')
agent = Agent(model=model,
model_settings=model_settings)

result = agent.run_sync('Open a new browser tab')
print(result.output)

```

## การอ้างอิงคำตอบก่อนหน้า

Responses API รองรับการอ้างอิงคำตอบก่อนหน้าของโมเดลผ่านพารามิเตอร์ `previous_response_id` เพื่อให้บริบทการสนทนาและ reasoning ก่อนหน้ายังคงอยู่ครบถ้วน

ใน Pydantic AI สามารถใช้งานได้ผ่านฟิลด์ `openai_previous_response_id` ใน `OpenAIResponsesModelSettings`

```

from pydantic_ai import Agent
from pydantic_ai.models.openai import
OpenAIResponsesModel, OpenAIResponsesModelSettings

model = OpenAIResponsesModel('gpt-5')
agent = Agent(model=model)

result = agent.run_sync('The secret is 1234')

model_settings = OpenAIResponsesModelSettings(
    openai_previous_response_id=result.all_messages()
    [-1].provider_response_id
)

result = agent.run_sync('What is the secret code?',
model_settings=model_settings)
print(result.output)
#> 1234

```

ด้วยวิธีนี้ โมเดลสามารถต่อยอดจาก reasoning เดิมของ  
 ตนเองได้ โดยไม่จำเป็นต้องส่ง message history ทั้งหมดซ้ำ  
 อีกครั้ง

## การอ้างอิงคำตอบก่อนหน้าแบบอัตโนมัติ

หากตั้งค่า `openai_previous_response_id` เป็น `'auto'` Pydantic AI จะเลือก `provider_response_id` ล่าสุดจาก message history ให้โดยอัตโนมัติ และจะตัดข้อความก่อนหน้านี้ ออกจากคำขอ เพื่อให้ OpenAI API ใช้ประโยชน์จาก server-side history ได้อย่างมีประสิทธิภาพมากขึ้น

```
from pydantic_ai import Agent
from pydantic_ai.models.openai import
OpenAIResponsesModel, OpenAIResponsesModelSettings

model = OpenAIResponsesModel('gpt-5')
agent = Agent(model=model)

result1 = agent.run_sync('Tell me a joke.')
print(result1.output)
#> Did you hear about the toothpaste scandal? They
called it Colgate.

# เมื่อกำหนดเป็น 'auto'
# ระบบจะส่งเฉพาะ provider_response_id ล่าสุด
# และข้อความหลังจากนั้นไปยัง OpenAI
model_settings = OpenAIResponsesModelSettings(
    openai_previous_response_id='auto'
)

result2 = agent.run_sync(
```

```
'Explain?',  
message_history=result1.new_messages(),  
model_settings=model_settings  
)  
print(result2.output)  
#> This is an excellent joke invented by Samuel  
Colvin, it needs no explanation.
```

แนวทางนี้ช่วยลดขนาดของ context ที่ต้องส่งซ้ำ และเพิ่มประสิทธิภาพในการเรียกใช้งาน Responses API โดยยังคงความต่อเนื่องของการสนทนาไว้ครบถ้วน

## Google

ติดตั้ง Google Model ดังนี้

```
pip install "pydantic-ai-slim[google]"
```

ตั้งค่า API Key ของ Gemini ใน Environment Variable  
ดังนี้

```
export GOOGLE_API_KEY=your-api-key
```

## Google Generative Language API

เชื่อมต่อกับโมเดลของ Google ทำได้ดังนี้

```
from pydantic_ai import Agent

agent = Agent('google-gla:gemini-2.5-pro')
...
```

หรือตั้งค่าผ่าน GoogleModel ดังนี้

```
from pydantic_ai import Agent
from pydantic_ai.models.google import GoogleModel
from pydantic_ai.providers.google import GoogleProvider

provider = GoogleProvider(api_key='your-api-key')
model = GoogleModel('gemini-2.5-flash',
                    provider=provider)
agent = Agent(model)
```

## Vertex AI

หากคุณใช้ Vertex AI ในบริการ Google Cloud เรียกใช้งานได้ดังนี้

```
from pydantic_ai import Agent

agent = Agent('google-vertex:gemini-2.5-pro')
...
```

หรือ

```
from pydantic_ai import Agent
from pydantic_ai.models.google import GoogleModel
from pydantic_ai.providers.google import GoogleProvider

provider = GoogleProvider(vertexai=True)
model = GoogleModel('gemini-2.5-flash',
provider=provider)
agent = Agent(model)
```

หากต้องการระบุ Service Account ผ่านไฟล์ JSON  
ทำได้ดังนี้

```
from google.oauth2 import service_account

from pydantic_ai import Agent
from pydantic_ai.models.google import GoogleModel
from pydantic_ai.providers.google import GoogleProvider
```

```

credentials =
service_account.Credentials.from_service_account_file
(
    'path/to/service-account.json',
    scopes=['https://www.googleapis.com/auth/cloud-
platform'],
)
provider = GoogleProvider(credentials=credentials)
model = GoogleModel('gemini-2.5-flash',
provider=provider)
agent = Agent(model)
...

```

หากต้องการระบุ location ที่ให้บริการ VertexAI ทำได้  
ดังนี้

```

from pydantic_ai import Agent
from pydantic_ai.models.google import GoogleModel
from pydantic_ai.providers.google import
GoogleProvider

provider = GoogleProvider(vertexai=True,
location='asia-east1', project='your-gcp-project-id')
model = GoogleModel('gemini-2.5-flash',
provider=provider)

```

```
agent = Agent(model)
...
```

นอกจากนี้ยังสามารถตั้งค่า Model เพิ่มเติมโดยใช้  
GoogleModelSettings ดังนี้

```
from google.genai.types import HarmBlockThreshold, HarmCategory

from pydantic_ai import Agent
from pydantic_ai.models.google import GoogleModel, GoogleModelSettings

settings = GoogleModelSettings(
    temperature=0.2,
    max_tokens=1024,
    google_thinking_config={'thinking_budget': 2048},
    google_safety_settings=[
        {
            'category':
HarmCategory.HARM_CATEGORY_HATE_SPEECH,
            'threshold':
HarmBlockThreshold.BLOCK_LOW_AND_ABOVE,
        }
    ]
)
```

```
model = GoogleModel('gemini-2.5-flash')
agent = Agent(model, model_settings=settings)
```

## Model Garden

คุณสามารถเข้าถึงโมเดลจาก **Model Garden** ที่รองรับ generateContent API และเปิดให้ใช้งานภายใต้โปรเจกต์ GCP ของคุณได้ ซึ่งรวมถึงโมเดลตระกูล Gemini และโมเดลอื่น ๆ อีกมากมาย โดยสามารถระบุ model\_name ได้ตามรูปแบบต่อไปนี้

- {model\_id} สำหรับโมเดล Gemini
- {publisher}/{model\_id}
- publishers/{publisher}/models/{model\_id}
- projects/{project}/locations/{location}/publishers/{publisher}/models/{model\_id}

รูปแบบเหล่านี้ช่วยให้คุณอ้างอิงโมเดลได้ทั้งจากผู้เผยแพร่ (publisher) โดยตรง หรือจากโมเดลที่ถูกผูกไว้กับโปรเจกต์และภูมิภาค (location) เฉพาะใน GCP

ตัวอย่างด้านล่างแสดงการใช้งานโมเดลจาก Model Garden ผ่าน GoogleModel โดยกำหนดค่าโปรเจกต์และภูมิภาคด้วย GoogleProvider

```
from pydantic_ai import Agent
from pydantic_ai.models.google import GoogleModel
from pydantic_ai.providers.google import GoogleProvider

provider = GoogleProvider(
    project='your-gcp-project-id',
    location='us-central1', # region ที่โมเดลพร้อมให้ใช้
    งาน
)

model = GoogleModel(
    'meta/llama-3.3-70b-instruct-maas',
    provider=provider
)

agent = Agent(model)
...
```

ในตัวอย่างนี้ เอเจนต์จะใช้โมเดล meta/llama-3.3-70b-instruct-maas จาก Model Garden ซึ่งถูกเรียกใช้งานผ่านโปรเจกต์และภูมิภาคที่กำหนดไว้ ทำให้สามารถผสานโมเดลจากผู้

ให้บริการหลายรายเข้ากับ Pydantic AI ได้อย่างยืดหยุ่นภายใต้โครงสร้างของ GCP

## การรับอินพุตแบบ Document, Image, Audio และ Video

GoogleModel รองรับอินพุตแบบ **Multi-modal** อย่างครบถ้วน ไม่ว่าจะเป็นเอกสาร รูปภาพ เสียง หรือวิดีโอ ทำให้สามารถส่งข้อมูลหลากหลายรูปแบบให้โมเดลประมวลผลรวมกันได้

## การใช้งาน YouTube Video URL โดยตรง

คุณสามารถส่ง URL ของวิดีโอ YouTube ให้กับโมเดล Google ได้โดยตรง เช่น

```
from pydantic_ai import Agent, VideoUrl
from pydantic_ai.models.google import GoogleModel

agent = Agent(GoogleModel('gemini-2.5-flash'))
result = agent.run_sync(
    [
        'What is this video about?',
        VideoUrl(url='https://www.youtube.com/watch?v=dQw4w9WgXcQ'),
    ],
)
```

```
]
)
print(result.output)
```

ในตัวอย่างนี้ เอเจนต์จะรับทั้งคำถามและ URL ของวิดีโอ เพื่อให้โมเดลวิเคราะห์เนื้อหาของวิดีโอและตอบกลับมาโดยตรง

## การอัปโหลดไฟล์ผ่าน Files API

นอกจาก URL ภายนอกแล้ว คุณยังสามารถอัปโหลดไฟล์ผ่าน **Files API** และส่งไฟล์นั้นให้โมเดลในรูปแบบ URL ได้เช่นกัน

```
from pydantic_ai import Agent, DocumentUrl
from pydantic_ai.models.google import GoogleModel
from pydantic_ai.providers.google import GoogleProvider

provider = GoogleProvider()
file = provider.client.files.upload(file='pydantic-ai-logo.png')
assert file.uri is not None

agent = Agent(GoogleModel('gemini-2.5-flash',
provider=provider))
```

```

result = agent.run_sync(
    [
        'What company is this logo from?',
        DocumentUrl(url=file.uri,
media_type=file.mime_type),
    ]
)
print(result.output)

```

ในกรณีนี้ ไฟล์รูปภาพจะถูกอัปโหลดก่อน จากนั้นนำ uri ของไฟล์มาใช้เป็นอินพุตให้กับเอเจนต์ พร้อมระบุชนิดสื่อ (media\_type) ให้ถูกต้อง

## การตั้งค่าโมเดล

คุณสามารถปรับพฤติกรรมของโมเดลได้ผ่าน GoogleModelSettings เพื่อควบคุมรูปแบบการตอบ ความยาวของผลลัพธ์ และการตั้งค่าด้านความปลอดภัย

```

from google.genai.types import HarmBlockThreshold, HarmCategory

from pydantic_ai import Agent
from pydantic_ai.models.google import GoogleModel, GoogleModelSettings

```

```

settings = GoogleModelSettings(
    temperature=0.2,
    max_tokens=1024,
    google_thinking_config={'thinking_level': 'low'},
    google_safety_settings=[
        {
            'category':
HarmCategory.HARM_CATEGORY_HATE_SPEECH,
            'threshold':
HarmBlockThreshold.BLOCK_LOW_AND_ABOVE,
        }
    ]
)

model = GoogleModel('gemini-2.5-pro')
agent = Agent(model, model_settings=settings)
...

```

การตั้งค่าเหล่านี้ช่วยให้คุณควบคุมระดับความสุ่มของคำตอบ (temperature) จำนวนโทเคนสูงสุด และระดับการ “คิด” ของโมเดล รวมถึงกำหนดกฎด้านความปลอดภัยได้อย่างละเอียด

## การปิด Thinking

สำหรับโมเดลที่ต่ำกว่า **Gemini 2.5 Pro** คุณสามารถปิดการทำงานด้าน thinking ได้ โดยตั้งค่า thinking\_budget เป็น 0

```
from pydantic_ai import Agent
from pydantic_ai.models.google import GoogleModel,
GoogleModelSettings

model_settings = GoogleModelSettings(
    google_thinking_config={'thinking_budget': 0}
)

model = GoogleModel('gemini-2.5-flash')
agent = Agent(model, model_settings=model_settings)
...
```

## การตั้งค่าความปลอดภัย

คุณสามารถกำหนดนโยบายด้านความปลอดภัยของโมเดลได้ผ่านฟิลด์ google\_safety\_settings เช่น การบล็อกเนื้อหาที่เข้าข่าย Hate Speech

```

from google.genai.types import HarmBlockThreshold,
HarmCategory

from pydantic_ai import Agent
from pydantic_ai.models.google import GoogleModel,
GoogleModelSettings

model_settings = GoogleModelSettings(
    google_safety_settings=[
        {
            'category':
HarmCategory.HARM_CATEGORY_HATE_SPEECH,
            'threshold':
HarmBlockThreshold.BLOCK_LOW_AND_ABOVE,
        }
    ]
)

model = GoogleModel('gemini-2.5-flash')
agent = Agent(model, model_settings=model_settings)
...

```

การตั้งค่านี้ช่วยให้คุณควบคุมระดับการกรองเนื้อหาได้ตามข้อกำหนดด้านจริยธรรมและความปลอดภัยของแอปพลิเคชันที่คุณพัฒนา

## Anthropic (Claude)

ติดตั้ง Anthropic Model ดังนี้

```
pip install "pydantic-ai-slim[anthropic]"
```

ตั้งค่า API Key ของ Anthropic ใน Environment Variable ดังนี้

```
export ANTHROPIC_API_KEY='your-api-key'
```

เรียกใช้โมเดล Claude ของ Anthropic ดังนี้

```
from pydantic_ai import Agent  
  
agent = Agent('anthropic:claude-4-5-sonnet-latest')
```

หรือเรียกใช้ผ่าน AnthropicModel ดังนี้

```
from pydantic_ai import Agent  
from pydantic_ai.models.anthropic import AnthropicModel  
  
model = AnthropicModel('claude-4-5-sonnet-latest')  
agent = Agent(model)
```

กำหนด API Key ผ่าน provider ดังนี้

```
from pydantic_ai import Agent
from pydantic_ai.models.anthropic import AnthropicModel
from pydantic_ai.providers.anthropic import AnthropicProvider

provider = AnthropicProvider(api_key='your-api-key')
model = AnthropicModel('claude-4-5-sonnet-latest',
provider=provider)
agent = Agent(model)
```

## การใช้งานร่วมกับ Cloud Platform

คุณสามารถใช้งานโมเดลของ Anthropic ผ่านแพลตฟอร์มคลาวด์ต่าง ๆ ได้ โดยการส่ง **custom client** เข้าไปให้กับ AnthropicProvider

### AWS Bedrock

หากต้องการใช้งานโมเดล Claude ผ่าน AWS Bedrock ให้ทำตามเอกสารของ Anthropic เพื่อสร้าง