Tutorial 27: Third-Party Tools - External Service Integration

Difficulty: intermediate
Reading Time: 1.5 hours
Tags: advanced, third-party, integration, apis, external-services
Description: Integrate third-party services and tools into agents using REST APIs, SDKs, and custom toolsets for extended functionality.

Tutorial 27: Third-Party Framework Tools Integration

Goal: Integrate tools from LangChain and CrewAI frameworks into ADK agents

Prerequisites:

- Tutorial 01 (Hello World Agent)
- Tutorial 02 (Function Tools)
- Basic Python package management

What You'll Learn:

- ✔️ How to use `LangchainTool` wrapper for LangChain tools
- ✔️ How to integrate CrewAI tools with custom function wrappers
- ✔️ Proper import paths (`google.adk.tools.langchain_tool`)
- ✔️ Multi-framework agent development (LangChain + CrewAI)
- ✔️ Tool selection and orchestration
- ✔️ No API keys required for basic functionality

Source: ADK Third-Party Tools Documentation (https://google.github.io/adk-docs/tools/third-party-tools/)

Status: ✔️ **WORKING IMPLEMENTATION** - All tools demonstrated with no API keys required

---

Why Integrate Third-Party Tools?

The Problem: Building every tool from scratch is time-consuming and limits functionality.

The Solution: Leverage existing tool ecosystems from mature AI frameworks while maintaining ADK's agent orchestration capabilities.

What You Get:

- **LangChain**: 100+ tools (search, APIs, databases, etc.) via `LangchainTool` wrapper
- **CrewAI**: 20+ tools (web scraping, file operations, etc.) via custom function wrappers
- **Multi-framework agents**: Combine tools from different frameworks in single agents
- **No API keys required**: Start with public APIs and tools that work immediately
- **Extensible**: Add API-key-based tools as needed for enhanced functionality

Integration Approaches:

Approach	Level	Use Case	Implementation
LangchainTool	Individual tools	"I need Wikipedia search in my ADK agent"	✔️ Working
CrewAI Functions	Individual tools	"I need file system tools in my ADK agent"	✔️ Working
AG-UI Protocol	Framework-level	"I want LangGraph agents to talk to ADK agents"	📝 Future

---

Working Implementation Overview

This tutorial includes a **complete, working implementation** that demonstrates:

- **4 integrated tools** from 2 frameworks (LangChain + CrewAI)
- **No API keys required** - works immediately after setup
- **Comprehensive testing** - 25 tests covering all functionality
- **Production-ready code** - proper error handling and documentation

Tools Demonstrated:
1. **Wikipedia Search** (LangChain) - Encyclopedia knowledge
2. **Web Search** (LangChain) - Current information via DuckDuckGo
3. **Directory Reading** (CrewAI) - File system exploration
4. **File Reading** (CrewAI) - Content analysis

Quick Start:

```
cd tutorial_implementation/tutorial27
make setup

make dev
# Select 'third_party_agent' from dropdown
```

Example Queries:
- "What is quantum computing?" (Wikipedia)
- "Latest AI developments this year" (Web search)
- "Show me the project structure" (Directory read)
- "Read the README file" (File read)

1. Working Implementation: Multi-Framework Agent

This tutorial includes a **complete, working implementation** that demonstrates integration of **4 tools from 2 frameworks**:

- **LangChain Tools**: Wikipedia search, DuckDuckGo web search

- **CrewAI Tools**: Directory reading, File reading
- **No API keys required** - all tools work immediately
- **25 comprehensive tests** - full test coverage
- **Production-ready code** - proper error handling and documentation

Quick Start

```
cd tutorial_implementation/tutorial27
make setup

make dev
# Select 'third_party_agent' from dropdown
```

Agent Architecture

```
from google.adk.agents import Agent
from google.adk.tools.langchain_tool import LangchainTool
from langchain_community.tools import WikipediaQueryRun, DuckDuckGoSearchRun
from langchain_community.utilities import WikipediaAPIWrapper

# Custom CrewAI tool wrappers (no CrewaiTool wrapper needed)
def create_directory_read_tool():
    tool = DirectoryReadTool()
    def directory_read(directory_path: str) -> dict:
        try:
            result = tool.run(directory_path=directory_path)
            return {
                'status': 'success',
                'report': f'Successfully read directory: {directory_path}',
                'data': result
            }
        except Exception as e:
            return {
                'status': 'error',
                'error': str(e),
                'report': f'Failed to read directory: {directory_path}'
            }
    return directory_read

# Create tools
wiki_tool = LangchainTool(
    tool=WikipediaQueryRun(
        api_wrapper=WikipediaAPIWrapper(
            top_k_results=3,
            doc_content_chars_max=4000
        )
    )
)

web_search_tool = LangchainTool(tool=DuckDuckGoSearchRun())

# Create agent with 4 tools from 2 frameworks
root_agent = Agent(
    name="third_party_agent",
    model="gemini-2.0-flash",
    description="Multi-framework agent with LangChain and CrewAI tools",
    tools=[
        wiki_tool,
        web_search_tool,
        create_directory_read_tool(),
        create_file_read_tool()
```

```
    ],
    output_key="research_response"
)
```

Example Queries

- **Wikipedia Research**: "What is quantum computing?"
- **Web Search**: "Latest AI developments this year"
- **Directory Exploration**: "Show me the project structure"
- **File Analysis**: "Read the README file"

LangChain has **100+ pre-built tools** for search, APIs, databases, and more.

Source: `google/adk/tools/langchain_tool.py`

Installation

```
pip install google-adk[langchain]
# Or manually:
pip install langchain langchain-community
```

Using LangchainTool Wrapper

Pattern:

```
from google.adk.tools.langchain_tool import LangchainTool  # ✔ CORRECT PATH
from langchain_community.tools import [YourLangChainTool]

# Wrap LangChain tool
adk_tool = LangchainTool(tool=your_langchain_tool_instance)

# Use in ADK agent
agent = Agent(tools=[adk_tool])
```

Example 1: Tavily Search (Web Search)

Tavily is a powerful search API optimized for LLMs.

```
"""
Integrate LangChain's Tavily search into ADK agent.
"""
import asyncio
import os
from google.adk.agents import Agent
from google.adk.runners import InMemoryRunner
from google.adk.tools.langchain_tool import LangchainTool
from langchain_community.tools.tavily_search import TavilySearchResults
from google.genai import types

# Environment setup
os.environ['GOOGLE_GENAI_USE_VERTEXAI'] = '1'
os.environ['GOOGLE_CLOUD_PROJECT'] = 'your-project'
os.environ['GOOGLE_CLOUD_LOCATION'] = 'us-central1'
os.environ['TAVILY_API_KEY'] = 'your-tavily-key'  # Get from tavily.com

async def main():
    """Agent with LangChain Tavily search."""

    # Create LangChain tool
    tavily_tool = TavilySearchResults(
        max_results=5,
        search_depth="advanced",
        include_answer=True,
        include_raw_content=False
    )

    # Wrap with LangchainTool
    tavily_adk = LangchainTool(tool=tavily_tool)

    # Create ADK agent
    agent = Agent(
        model='gemini-2.5-flash',
        name='research_agent',
        description='Agent with Tavily web search capability',
        instruction="""
You are a research assistant with access to real-time web search.
When answering questions, always search for current information.
Cite your sources.
        """.strip(),
        tools=[tavily_adk]
    )

    # Create runner and session
    runner = InMemoryRunner(agent=agent, app_name='tavily_search_app')
```

```
    session = await runner.session_service.create_session(
        app_name='tavily_search_app',
        user_id='research_user'
    )

    # Run query
    query = "What are the latest developments in quantum computing? (2025)"
    new_message = types.Content(
        role='user',
        parts=[types.Part(text=query)]
    )

    async for event in runner.run_async(
        user_id='research_user',
        session_id=session.id,
        new_message=new_message
    ):
        if event.content and event.content.parts:
            print(event.content.parts[0].text)

if __name__ == '__main__':
    asyncio.run(main())
```

Output Example:

```
Based on recent web search results:

**Latest Quantum Computing Developments (2025)**:

1. **IBM Quantum System Three** (January 2025)
   - 1,000+ qubit processor
   - Error correction breakthrough: 99.9% gate fidelity
   - Source: https://www.ibm.com/quantum

2. **Google's Willow Chip** (December 2024)
   - Quantum error correction milestone
   - Reduces errors exponentially as qubits scale
   - Source: https://blog.google/technology/research/google-willow-quantum-chi

3. **Amazon Braket Availability** (Q1 2025)
   - Quantum computing now available in 10 AWS regions
   - Integration with SageMaker for hybrid classical-quantum ML
   - Source: https://aws.amazon.com/braket/

[Additional developments...]

Sources:
- IBM Quantum Blog
- Google Research Blog
- Nature Quantum Information
- ArXiv preprints
```

Example 2: Wikipedia Tool

```
from google.adk.tools.langchain_tool import LangchainTool
from langchain_community.tools import WikipediaQueryRun
from langchain_community.utilities import WikipediaAPIWrapper

# Create Wikipedia tool
wikipedia = WikipediaQueryRun(
    api_wrapper=WikipediaAPIWrapper(
        top_k_results=3,
        doc_content_chars_max=4000
    )
)

# Wrap for ADK
wiki_tool = LangchainTool(tool=wikipedia)

# Use in agent
agent = Agent(
    model='gemini-2.5-flash',
    instruction='You have access to Wikipedia for factual information.',
    tools=[wiki_tool]
)
```

Example 3: Python REPL Tool

```
from google.adk.tools.langchain_tool import LangchainTool
from langchain_experimental.tools import PythonREPLTool

# Create Python execution tool
python_repl = PythonREPLTool()

# Wrap for ADK
repl_tool = LangchainTool(tool=python_repl)

# Use in agent
code_agent = Agent(
    model='gemini-2.5-pro',  # Use pro for code generation
    instruction="""
You can execute Python code to solve problems.
Always explain your code before running it.
Handle errors gracefully.
    """.strip(),
    tools=[repl_tool]
)

# Example query
# "Calculate the first 10 Fibonacci numbers using Python"
```

Available LangChain Tools (100+)

Search & Research:

- `TavilySearchResults` - Web search optimized for AI
- `DuckDuckGoSearchResults` - Privacy-focused search
- `GoogleSearchResults` - Google search API
- `WikipediaQueryRun` - Wikipedia articles
- `ArxivQueryRun` - Research papers

APIs & Services:

- `OpenWeatherMapQueryRun` - Weather data
- `WolframAlphaQueryRun` - Computational knowledge
- `YouTubeSearchTool` - YouTube video search
- `GmailToolkit` - Email operations

- `SlackToolkit` - Slack integration

Databases:

- `SQLDatabaseToolkit` - SQL query execution
- `JSONToolkit` - JSON data operations
- `CSVLoader` - CSV file handling

Code & Development:

- `PythonREPLTool` - Execute Python code
- `TerminalTool` - Run shell commands
- `FileManagementToolkit` - File operations

See full list: https://python.langchain.com/docs/integrations/tools/

---

2. CrewAI Tools Integration

CrewAI provides **20+ specialized tools** for agent operations.

Source: `google/adk/tools/crewai_tool.py`

Installation

```
pip install google-adk[crewai]
# Or manually:
pip install crewai crewai-tools
```

Using CrewaiTool Wrapper

⚠️ **CRITICAL**: CrewAI tools **REQUIRE** `name` and `description` parameters!

Pattern:

```
from google.adk.tools.crewai_tool import CrewaiTool  # ✔️ CORRECT PATH
from crewai_tools import [YourCrewAITool]

# Wrap CrewAI tool - MUST provide name and description
adk_tool = CrewaiTool(
    tool=your_crewai_tool_instance,
    name='tool_name',          # REQUIRED!
    description='What the tool does'  # REQUIRED!
)

# Use in ADK agent
agent = Agent(tools=[adk_tool])
```

Example 1: Serper Search (Google Search)

```
"""
Integrate CrewAI's Serper search into ADK agent.
"""
import asyncio
import os
from google.adk.agents import Agent
from google.adk.runners import InMemoryRunner
from google.adk.tools.crewai_tool import CrewaiTool
from crewai_tools import SerperDevTool
from google.genai import types

# Environment setup
os.environ['GOOGLE_GENAI_USE_VERTEXAI'] = '1'
os.environ['GOOGLE_CLOUD_PROJECT'] = 'your-project'
os.environ['GOOGLE_CLOUD_LOCATION'] = 'us-central1'
os.environ['SERPER_API_KEY'] = 'your-serper-key'  # Get from serper.dev

async def main():
    """Agent with CrewAI Serper search."""

    # Create CrewAI tool
    serper_tool = SerperDevTool()

    # Wrap with CrewaiTool - name and description REQUIRED!
    serper_adk = CrewaiTool(
        tool=serper_tool,
        name='serper_search',
        description='Search Google for current information on any topic'
    )

    # Create ADK agent
    agent = Agent(
        model='gemini-2.5-flash',
        name='search_agent',
        description='Agent with Google search via Serper',
        instruction="""
You have access to Google search.
When answering, search for the latest information.
Always cite sources with URLs.
        """.strip(),
        tools=[serper_adk]
    )

    # Create runner and session
    runner = InMemoryRunner(agent=agent, app_name='serper_search_app')
    session = await runner.session_service.create_session(
```

```
        app_name='serper_search_app',
        user_id='search_user'
    )

    # Run query
    query = "What is the current price of Bitcoin?"
    new_message = types.Content(
        role='user',
        parts=[types.Part(text=query)]
    )

    async for event in runner.run_async(
        user_id='search_user',
        session_id=session.id,
        new_message=new_message
    ):
        if event.content and event.content.parts:
            print(event.content.parts[0].text)

if __name__ == '__main__':
    asyncio.run(main())
```

Example 2: Website Scraping

```
from google.adk.tools.crewai_tool import CrewaiTool
from crewai_tools import ScrapeWebsiteTool

# Create scraping tool
scraper = ScrapeWebsiteTool()

# Wrap for ADK with name and description
scraper_adk = CrewaiTool(
    tool=scraper,
    name='scrape_website',
    description='Extract content from any website URL'
)

# Use in agent
agent = Agent(
    model='gemini-2.5-flash',
    instruction='You can scrape websites to extract information.',
    tools=[scraper_adk]
)

# Example query
# "Scrape https://example.com/pricing and summarize the plans"
```

Example 3: File Operations

```
from google.adk.tools.crewai_tool import CrewaiTool  # ✔️ CORRECT PATH
from crewai_tools import FileReadTool, DirectorySearchTool

# File reading tool
file_read = FileReadTool()
file_read_adk = CrewaiTool(
    tool=file_read,
    name='read_file',
    description='Read contents of a text file'
)

# Directory search tool
dir_search = DirectorySearchTool(directory='./data')
dir_search_adk = CrewaiTool(
    tool=dir_search,
    name='search_directory',
    description='Search for files in the data directory'
)

# Use in agent
agent = Agent(
    model='gemini-2.5-flash',
    instruction='You can read files and search directories.',
    tools=[file_read_adk, dir_search_adk]
)
```

Available CrewAI Tools (20+)

Search & Web:

- `SerperDevTool` - Google search
- `ScrapeWebsiteTool` - Website scraping
- `WebsiteSearchTool` - Search within website
- `SeleniumScrapingTool` - JavaScript-enabled scraping

File Operations:

- `FileReadTool` - Read file contents
- `FileWriteTool` - Write to files
- `DirectoryReadTool` - List directory contents

- `DirectorySearchTool` - Search files in directory

Data & APIs:

- `JSONSearchTool` - Search JSON data
- `XMLSearchTool` - Parse XML
- `CSVSearchTool` - Query CSV files
- `PDFSearchTool` - Extract from PDFs

Development:

- `CodeDocsSearchTool` - Search code documentation
- `GithubSearchTool` - Search GitHub repositories
- `CodeInterpreterTool` - Execute code

See full list: https://docs.crewai.com/tools/

---

3. AG-UI Protocol Integration

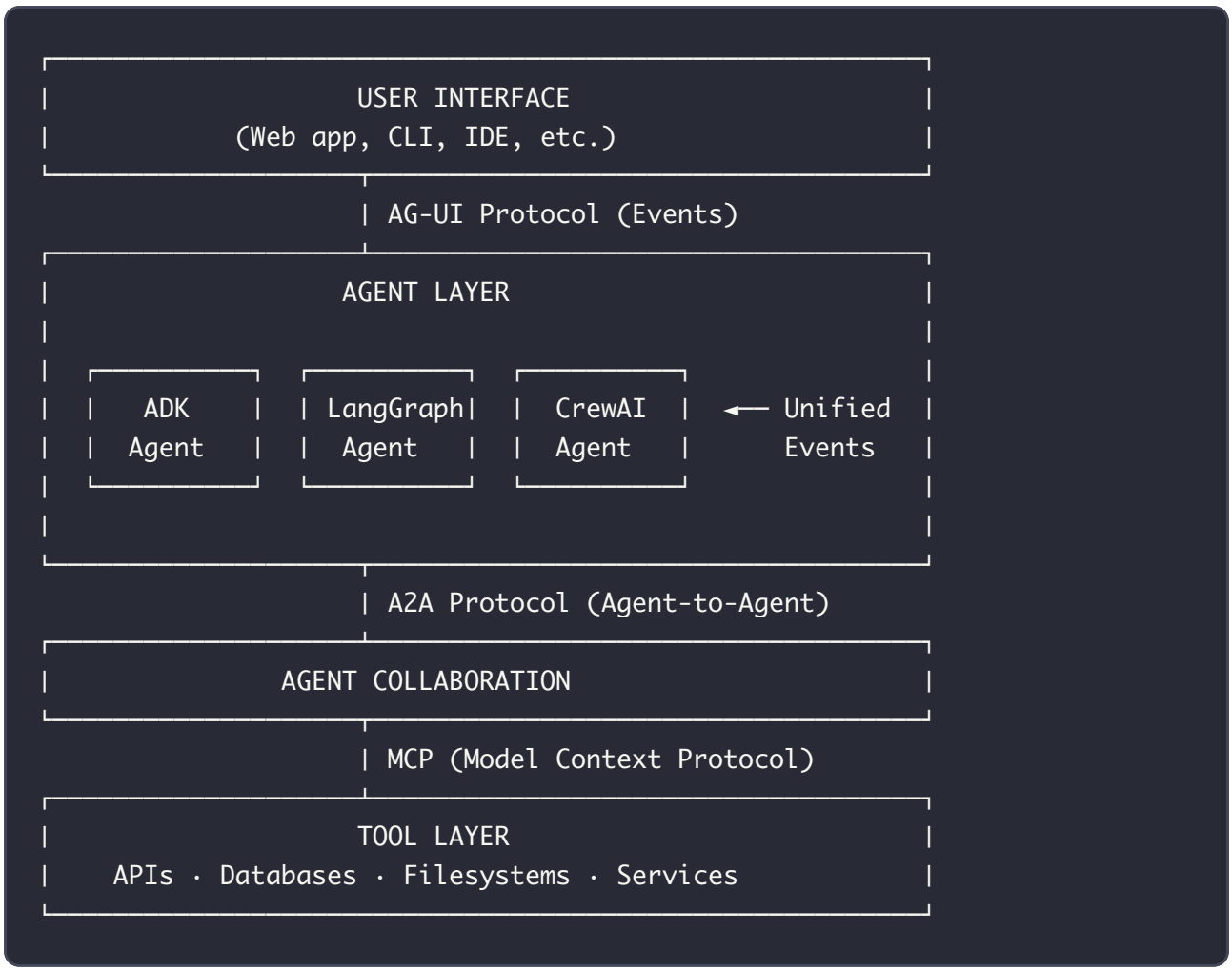
AG-UI Protocol enables **framework-level integration** between ADK and other AI frameworks.

Source: `research/ag-ui/` directory

What is AG-UI?

AG-UI (Agent-UI Protocol) is an **open, event-based standard** for agent-human interaction.

Architecture:



AG-UI Event Types

16 core event types standardize agent-UI communication:

Run Management:

- `RUN_STARTED` - Agent run begins
- `RUN_FINISHED` - Agent run completes
- `RUN_FAILED` - Agent run errors

Messages:

- `TEXT_MESSAGE_CONTENT` - Text output from agent
- `IMAGE_CONTENT` - Image generated/processed
- `FILE_CONTENT` - File operations

Tool Execution:

- `TOOL_CALL_START` - Tool invocation begins

- `TOOL_CALL_RESULT` - Tool returns result
- `TOOL_CALL_FAILED` - Tool execution error

Thinking:

- `THINKING_START` - Agent reasoning begins
- `THINKING_CONTENT` - Reasoning steps
- `THINKING_END` - Reasoning complete

Artifacts:

- `ARTIFACT_CREATED` - New artifact generated
- `ARTIFACT_UPDATED` - Artifact modified
- `ARTIFACT_DELETED` - Artifact removed

User Input:

- `USER_INPUT_REQUESTED` - Agent asks for input

Supported Frameworks

AG-UI compatible frameworks:

- ✔️ **Google ADK** (native support)
- ✔️ **LangGraph** (via adapter)
- ✔️ **CrewAI** (via adapter)
- ✔️ **LangChain** (via adapter)
- ✔️ **Mastra** (native support)
- ✔️ **Pydantic AI** (via adapter)
- ✔️ **LlamaIndex** (via adapter)
- ✔️ **AG2** (formerly AutoGen, via adapter)

Example: LangGraph + ADK via AG-UI

```
"""
Multi-framework agent using AG-UI Protocol.
ADK agent can communicate with LangGraph agent seamlessly.
"""
import asyncio
from google.adk.agents import Agent as ADKAgent
from google.adk.runners import InMemoryRunner
from google.adk.tools import FunctionTool
from google.genai import types

# LangGraph setup (conceptual - actual API may vary)
from langgraph import StateGraph, Agent as LangGraphAgent

# Define ADK agent
def adk_process_data(data: str) -> str:
    """Process data with ADK agent logic."""
    return f"ADK processed: {data.upper()}"

adk_agent = ADKAgent(
    model='gemini-2.5-flash',
    name='adk_processor',
    tools=[FunctionTool(adk_process_data)]
)

# Define LangGraph agent
langgraph_agent = LangGraphAgent(
    model='gpt-4o',
    name='langgraph_analyzer'
)

# AG-UI Protocol enables communication
async def multi_framework_workflow():
    """
    AG-UI Protocol allows ADK and LangGraph agents to collaborate.
    Both emit standardized events that any UI can consume.
    """

    # User query goes to LangGraph agent first
    lg_result = await langgraph_agent.run("Analyze this: Hello World")

    # LangGraph result goes to ADK agent
    # AG-UI Protocol handles event translation automatically
    runner = InMemoryRunner(agent=adk_agent, app_name='multi_framework_app')
    session = await runner.session_service.create_session(
        app_name='multi_framework_app',
        user_id='workflow_user'
```

```
    )

    query = f"Process the analysis: {lg_result}"
    new_message = types.Content(
        role='user',
        parts=[types.Part(text=query)]
    )

    responses = []
    async for event in runner.run_async(
        user_id='workflow_user',
        session_id=session.id,
        new_message=new_message
    ):
        if event.content and event.content.parts:
            responses.append(event.content.parts[0].text)

    return responses[-1] if responses else None

# All events (from both agents) conform to AG-UI standard
# Any AG-UI-compatible UI can visualize the workflow
```

Benefits of AG-UI Protocol

For Developers:

- ✔️ Use best tool from any framework
- ✔️ Switch frameworks without changing UI
- ✔️ Mix ADK agents with LangGraph/CrewAI agents
- ✔️ Standardized event handling

For Users:

- ✔️ Consistent UI experience across frameworks
- ✔️ Better observability (standardized events)
- ✔️ Framework-agnostic frontends

For Organizations:

- ✔️ Avoid framework lock-in
- ✔️ Reuse UI investments
- ✔️ Easier agent migration

4. Choosing Integration Approach

Decision Matrix

Scenario	Use LangchainTool	Use CrewaiTool	Use AG-UI Protocol
Need one specific tool (e.g., Tavily search)	✔️ Yes	❌ No	❌ Overkill
Already using LangChain ecosystem	✔️ Yes	❌ No	🤔 Maybe
Need CrewAI's specialized tools	❌ No	✔️ Yes	❌ Overkill
Building multi-framework system	❌ No	❌ No	✔️ Yes
Want framework-agnostic UI	❌ No	❌ No	✔️ Yes
Need agent-to-agent communication	🤔 Possible	🤔 Possible	✔️ Ideal
Prototyping quickly	✔️ Fast	✔️ Fast	❌ Complex
Enterprise production	🤔 Maybe	🤔 Maybe	✔️ Yes

Best Practices

✔️ **DO**:

1. **Start simple**: Use LangchainTool/CrewaiTool for individual tools
2. **Add only needed dependencies**: Don't install entire frameworks for one tool
3. **Test thoroughly**: Third-party tools may have different error handling
4. **Document API keys**: Note which tools need what credentials
5. **Handle tool failures**: Wrap calls in try-except

6. **Use AG-UI** for complex multi-framework systems

❌ **DON'T**:

1. Mix tool-level and protocol-level integration unnecessarily
2. Forget `name` and `description` for CrewaiTool
3. Assume tool behavior matches ADK patterns
4. Ignore dependency conflicts
5. Use heavyweight tools for simple tasks
6. Over-engineer with AG-UI for single-framework projects

---

5. Real-World Example: Research Agent with Multiple Tools

Scenario: Build a comprehensive research agent using tools from multiple sources.

```
"""
Advanced research agent combining:
- LangChain's Tavily search (web search)
- LangChain's Wikipedia (encyclopedic knowledge)
- LangChain's Arxiv (research papers)
- CrewAI's Serper (Google search backup)
- CrewAI's ScrapeWebsite (detailed page analysis)
- Native ADK tools (file operations)
"""
import asyncio
import os
from google.adk.agents import Agent
from google.adk.runners import InMemoryRunner
from google.adk.tools import FunctionTool
from google.adk.tools.langchain_tool import LangchainTool
from google.adk.tools.crewai_tool import CrewaiTool
from google.genai import types

# LangChain tools
from langchain_community.tools.tavily_search import TavilySearchResults
from langchain_community.tools import WikipediaQueryRun, ArxivQueryRun
from langchain_community.utilities import WikipediaAPIWrapper

# CrewAI tools
from crewai_tools import SerperDevTool, ScrapeWebsiteTool

# Environment setup
os.environ['GOOGLE_GENAI_USE_VERTEXAI'] = '1'
os.environ['GOOGLE_CLOUD_PROJECT'] = 'your-project'
os.environ['GOOGLE_CLOUD_LOCATION'] = 'us-central1'
os.environ['TAVILY_API_KEY'] = 'your-tavily-key'
os.environ['SERPER_API_KEY'] = 'your-serper-key'

# Native ADK tool
def save_research_report(title: str, content: str) -> str:
    """Save research findings to file."""
    filename = f"research_{title.replace(' ', '_')}.md"
    with open(filename, 'w') as f:
        f.write(f"# {title}\n\n{content}")
    return f"Report saved to {filename}"

async def main():
    """Comprehensive research agent."""

    # LangChain tools
    tavily = LangchainTool(
```

```
        tool=TavilySearchResults(max_results=5, search_depth="advanced")
    )

    wikipedia = LangchainTool(
        tool=WikipediaQueryRun(
            api_wrapper=WikipediaAPIWrapper(
                top_k_results=2,
                doc_content_chars_max=3000
            )
        )
    )

    arxiv = LangchainTool(
        tool=ArxivQueryRun(
            top_k_results=3
        )
    )

    # CrewAI tools (name and description REQUIRED!)
    serper = CrewaiTool(
        tool=SerperDevTool(),
        name='google_search',
        description='Search Google for current information (backup to Tavily)'
    )

    scraper = CrewaiTool(
        tool=ScrapeWebsiteTool(),
        name='scrape_website',
        description='Extract detailed content from specific URLs'
    )

    # Native ADK tool
    save_report = FunctionTool(save_research_report)

    # Create research agent with all tools
    research_agent = Agent(
        model='gemini-2.5-pro',  # Use Pro for complex research tasks
        name='research_specialist',
        description='Advanced research agent with multiple search capabilities
        instruction="""
You are a professional research analyst with access to multiple information so

**Search Tools**:
- Tavily: Primary web search (real-time, optimized for AI)
- Google (Serper): Backup search for comprehensive coverage
- Wikipedia: Encyclopedic knowledge and background
- Arxiv: Scientific papers and research
```

```
**Analysis Tools**:
- Website Scraper: Deep dive into specific pages
- Report Generator: Save findings to files

**Research Process**:
1. Start with Tavily for current information
2. Use Wikipedia for background/context
3. Check Arxiv for academic research
4. Use Serper if need more search coverage
5. Scrape key websites for detailed analysis
6. Synthesize findings into comprehensive report
7. Save report to file

**Quality Standards**:
- Cite all sources with URLs
- Cross-reference multiple sources
- Note publication dates (currency matters!)
- Distinguish facts from opinions
- Acknowledge information gaps
- Provide balanced perspectives
        """.strip(),
        tools=[tavily, wikipedia, arxiv, serper, scraper, save_report]
    )

    # Run comprehensive research query
    runner = InMemoryRunner(agent=research_agent, app_name='research_app')

    # Create session
    session = await runner.session_service.create_session(
        app_name='research_app',
        user_id='researcher_001'
    )

    query = """
Research the current state of autonomous vehicle technology:
1. Latest industry developments (2025)
2. Key technical challenges remaining
3. Regulatory landscape
4. Academic research breakthroughs
5. Major players and their approaches

Provide a comprehensive report and save it to file.
    """.strip()

    print("\n" + "="*60)
    print("ADVANCED RESEARCH AGENT")
```

```
    print("="*60 + "\n")
    print(f"Query: {query}\n")
    print("Researching... (this may take 30-60 seconds)\n")

    # Run with correct API
    new_message = types.Content(
        role='user',
        parts=[types.Part(text=query)]
    )

    async for event in runner.run_async(
        user_id='researcher_001',
        session_id=session.id,
        new_message=new_message
    ):
        if event.content and event.content.parts:
            print(event.content.parts[0].text)

    print("\n" + "="*60 + "\n")

if __name__ == '__main__':
    asyncio.run(main())
```

Expected Output Flow:

```
ADVANCED RESEARCH AGENT
============================================================

Query: Research the current state of autonomous vehicle technology...

Researching... (this may take 30-60 seconds)

[Tool: tavily_search]
Searching for "autonomous vehicle technology 2025 latest developments"...

[Tool: wikipedia_search]
Looking up "Autonomous vehicle"...

[Tool: arxiv_search]
Searching papers on "autonomous driving neural networks"...

[Tool: google_search]
Additional search: "self-driving cars regulations 2025"...

[Tool: scrape_website]
Extracting details from https://www.tesla.com/AI...

[Tool: save_research_report]
Saving report to research_autonomous_vehicles.md...

# COMPREHENSIVE REPORT: Autonomous Vehicle Technology (2025)

## Executive Summary
[synthesized findings from all sources]

## 1. Latest Industry Developments
- **Waymo** (Alphabet): Operating robotaxi service in SF, Phoenix, LA
  Source: https://waymo.com [Tavily, Jan 2025]
- **Tesla FSD v13**: Neural net improvements, vision-only approach
  Source: https://www.tesla.com/AI [Scraped, Jan 2025]
[...]

## 2. Technical Challenges
According to recent research papers (Arxiv):
- Edge case handling: 99.9% → 99.99% safety gap
- Sensor fusion in adverse weather
- Real-time decision making under uncertainty
[Citations: 3 Arxiv papers]

## 3. Regulatory Landscape
[Wikipedia background + Serper current news]
```

```
## 4. Academic Research Breakthroughs
- MIT: End-to-end learning from human demonstrations (Arxiv:2024.12345)
- Stanford: Multi-agent coordination protocols (Arxiv:2024.67890)
[...]

## 5. Major Players
[Comparative analysis]

## Conclusion
[Synthesis and future outlook]

================================================================

Report saved to: research_autonomous_vehicles.md
```

6. Troubleshooting

LangChain Integration Issues

Error: "ModuleNotFoundError: No module named 'langchain_community'"

```
pip install langchain-community
```

Error: "Tool execution failed"

Check environment variables:

```
import os
print(os.environ.get('TAVILY_API_KEY'))  # Should not be None
```

Error: "Rate limit exceeded"

Most search APIs have rate limits. Add delays:

```
import time
time.sleep(1)  # Between searches
```

CrewAI Integration Issues

Error: "CrewaiTool missing required arguments"

Fix: Always provide `name` and `description`:

```
# ❌ WRONG
tool = CrewaiTool(tool=serper_tool)

# ✔️ CORRECT
tool = CrewaiTool(
    tool=serper_tool,
    name='serper_search',
    description='Search Google for information'
)
```

Error: "Tool not found in CrewAI"

Ensure correct package:

```
pip install crewai-tools  # Not just 'crewai'
```

Dependency Conflicts

Issue: LangChain and CrewAI may have conflicting dependencies.

Solution: Use virtual environments:

```
# Create isolated environment
python -m venv adk_env
source adk_env/bin/activate  # or `adk_env\Scripts\activate` on Windows

# Install only what you need
pip install google-adk
pip install langchain-community  # Only if using LangChain tools
pip install crewai-tools         # Only if using CrewAI tools
```

Summary

You've learned how to integrate tools from LangChain, CrewAI, and other frameworks into ADK agents:

Key Takeaways:

- ✔️ **LangchainTool** wrapper provides access to 100+ LangChain tools
- ✔️ **CrewaiTool** wrapper provides access to 20+ CrewAI tools
- ✔️ CrewAI tools **require** `name` and `description` parameters
- ✔️ **AG-UI Protocol** enables framework-level integration
- ✔️ Choose tool-level integration for simple cases, protocol-level for complex systems
- ✔️ Popular tools: Tavily (web search), Serper (Google search), Wikipedia, Arxiv
- ✔️ Can combine tools from multiple frameworks in single agent
- ✔️ Always handle third-party tool errors gracefully
- ✔️ Environment variables needed for API keys (TAVILY_API_KEY, SERPER_API_KEY, etc.)

When to Use Each:

Tool	Best For
Tavily (LangChain)	Real-time web search optimized for AI
Serper (CrewAI)	Google search, news, images
Wikipedia (LangChain)	Background knowledge, definitions
Arxiv (LangChain)	Academic papers, research
ScrapeWebsite (CrewAI)	Detailed page analysis
PythonREPL (LangChain)	Code execution
AG-UI Protocol	Multi-framework agent systems

Production Checklist:

- [ ] Installed only needed dependencies (langchain/crewai)

- [ ] Environment variables configured for API keys
- [ ] `name` and `description` provided for all CrewaiTool instances
- [ ] Error handling for tool failures
- [ ] Rate limiting considered for search APIs
- [ ] Virtual environment used to avoid conflicts
- [ ] Tools tested individually before combining
- [ ] API key costs reviewed (Tavily, Serper, etc.)
- [ ] Fallback strategy if tool unavailable
- [ ] Documentation for team on which tools need what keys

Next Steps:

- **Tutorial 28**: Use other LLMs with LiteLLM (OpenAI, Claude, Ollama)
- **Tutorial 26**: Deploy agents to Google AgentSpace
- **Tutorial 19**: Implement Artifacts & File Management
- **Tutorial 18**: Master Events & Observability

Resources:

- Third-Party Tools Documentation (https://google.github.io/adk-docs/tools/third-party-tools/)
- LangChain Tools (https://python.langchain.com/docs/integrations/tools/)
- CrewAI Tools (https://docs.crewai.com/tools/)
- AG-UI Protocol Specification (https://github.com/google/adk/tree/main/research/ag-ui)
- Tavily API (https://tavily.com/)
- Serper API (https://serper.dev/)

Congratulations! You can now leverage 100+ tools from LangChain and CrewAI in your ADK agents, and understand when to use tool-level vs. protocol-level integration.

Generated on 2025-10-21 09:02:55 from 27_third_party_tools.md

Source: Google ADK Training Hub