Claude Agents & Sub-Agents

Technical Architecture for Data Management

A comprehensive technical deep-dive into Anthropic's agentic AI framework, with practical applications for enterprise data foundations, data quality automation, and modern data management lifecycles.

Author: Aman Patel

Date: January 2026

Version: 1.0

Table of Contents

1. Executive Summary

The emergence of agentic AI represents a paradigm shift in how we approach automation within enterprise data management. Anthropic's Claude Code, and its underlying Claude Agent SDK, introduces a powerful framework for building specialized AI agents that operate with isolated context windows, domain-specific tooling, and orchestrated workflows.

This technical report provides a comprehensive analysis of Claude's sub-agent architecture, with a specific focus on applications within data foundations, data quality automation, and the modern data management lifecycle. The analysis draws from Anthropic's official documentation, community implementations, and emerging research in multi-agent AI systems.

Key Findings

- **Context Isolation:** Sub-agents operate with independent context windows, preventing cross-contamination and enabling parallel execution of specialized tasks.
- **90%+ Performance Gains:** Anthropic's internal testing shows multi-agent architectures outperform single-agent approaches by over 90%, at the cost of ~15x token consumption.
- **Nascent Data Management Ecosystem:** While the Claude Code sub-agent ecosystem is rapidly growing for software engineering, data-specific implementations remain an untapped opportunity.
- **MCP Integration:** The Model Context Protocol enables sub-agents to connect to databases, cloud platforms, and external tools, making enterprise data integration viable.
- **Productisation Potential:** Sub-agents can be packaged as plugins, enabling consulting practices to distribute standardised data management frameworks across client engagements.

Strategic Implications

For data leaders and practitioners, the sub-agent paradigm offers a compelling model for automating repetitive data engineering tasks while maintaining human oversight on strategic decisions. The ability to define specialized agents for data profiling, rule generation, cataloguing, and governance creates opportunities for significant productivity gains across the data lifecycle.

However, practitioners must carefully consider token economics, context management strategies, and the appropriate balance between automation and human judgment—particularly in areas requiring domain expertise or regulatory compliance.

2. Introduction to Claude Agents

2.1 What Are Claude Agents?

Claude agents represent a fundamental evolution in how AI assistants operate. Unlike traditional conversational AI that responds to individual prompts, agents operate in continuous feedback loops: gathering context, taking action, verifying results, and iterating until objectives are achieved.

The key innovation is giving Claude access to a computer—specifically, the ability to execute bash commands, read and write files, search codebases, and interact with external systems. This transforms Claude from a conversational assistant into an autonomous worker capable of completing multi-step tasks with minimal human intervention.

> *"The key design principle behind the Claude Agent SDK is to give your agents a computer, allowing them to work like humans do."*
>
> *— Anthropic Engineering Blog, 2025*

Core Agent Capabilities

Capability	Description	Data Management Application
File System Access	Read, write, edit files directly	Generate DDL scripts, dbt models, documentation
Bash Execution	Run shell commands and scripts	Execute SQL queries, run profiling tools
Context Management	Maintain state across operations	Track data lineage, accumulate profiling results
Tool Invocation	Call external APIs and services	Connect to databases, cloud platforms, catalogs
Verification Loops	Validate outputs and iterate	Test data quality rules, verify transformations

2.2 The Claude Agent SDK

In 2025, Anthropic renamed the Claude Code SDK to the Claude Agent SDK, reflecting its broader application beyond software development. The SDK provides primitives for building agents across any workflow: finance agents for portfolio analysis, personal assistants for calendar management, customer support agents for ticket resolution, and—critically for this analysis—data engineering agents for pipeline automation.

The SDK's core components include:

- **Tool Definitions:** Structured interfaces for Claude to interact with external systems
- **Context Compaction:** Automatic summarization when approaching context limits
- **Sub-agent Orchestration:** Delegation of specialized tasks to isolated agent instances
- **MCP Integration:** Standardized protocol for connecting to databases, APIs, and services
- **Resumable Sessions:** Ability to pause and resume long-running agent tasks

2.3 Multi-Agent Architecture

Anthropic's research on multi-agent systems demonstrates significant performance advantages over single-agent approaches. In their Claude Research agent implementation, a lead orchestrator (Claude Opus 4) coordinates specialized sub-agents (Claude Sonnet 4) to search, analyse, and synthesize information in parallel.

Metric	Single Agent	Multi-Agent	Improvement
Task Completion Rate	Baseline	+90.2%	Significant
Token Consumption	1x	~15x	Higher cost
Parallelization	Sequential	Parallel	Faster execution
Context Pollution	High risk	Isolated	Cleaner context

Source: Anthropic Engineering Blog, 'Building agents with the Claude Agent SDK', 2025

3. Anatomy of Sub-Agents

This section provides a detailed technical breakdown of Claude Code sub-agent architecture, covering file structure, configuration options, and implementation patterns.

3.1 File Structure & Priority

Sub-agents are defined as Markdown files with YAML frontmatter, stored in specific directories that determine their scope and priority:

Directory Structure

```
your-project/
■■■ .claude/
■   ■■■ agents/                    # Project-level agents (HIGHEST priority)
■   ■   ■■■ dq-profiler.md
■   ■   ■■■ dq-recommender.md
■   ■   ■■■ data-modeller.md
■   ■■■ settings.local.json
■
■■■ ~/.claude/
■   ■■■ agents/                    # User-level agents (across all projects)
■       ■■■ global-reviewer.md
■
■■■ .mcp.json                      # MCP server configurations
■■■ CLAUDE.md                      # Project context and instructions
```

Priority	Location	Scope	Use Case
1 (Highest)	.claude/agents/	Current project only	Project-specific workflows
2	CLI --agents flag	Current session	Testing, automation scripts
3 (Lowest)	~/.claude/agents/	All projects	Personal utilities, global tools

3.2 YAML Frontmatter Configuration

Every sub-agent file begins with YAML frontmatter that defines its configuration. This metadata controls when the agent is invoked, what tools it can access, and how it behaves.

YAML Frontmatter Example

```
---
name: dq-profiler
description: Data Quality Profiler. Use PROACTIVELY when analysing datasets.
tools: Read, Bash, Glob, Grep
model: sonnet
permissionMode: default
skills: data-profiling, sql-analysis
---

# System prompt content begins here...
```

Configuration Field Reference

Field	Required	Type	Description
name	Yes	string	Unique identifier (lowercase, hyphens). Example: dq-profiler
description	Yes	string	Natural language description. Claude uses this to decide delegation.
tools	No	string	Comma-separated tool list. Omit to inherit ALL tools.
model	No	string	sonnet | opus | haiku | inherit. Defaults to sonnet.
permissionMode	No	string	default | acceptEdits | bypassPermissions | plan | ignore
skills	No	string	Comma-separated skills to auto-load on agent start.

Critical: The description Field

The description field is the primary mechanism Claude uses to determine when to delegate tasks. Writing effective descriptions is crucial for reliable agent invocation:

- Include action verbs: 'Profile', 'Analyse', 'Generate', 'Review'
- Use 'PROACTIVELY' or 'MUST BE USED' for aggressive delegation
- Specify trigger contexts: 'after code changes', 'when analysing datasets'
- Keep descriptions concise but specific (1-2 sentences)

3.3 System Prompt Engineering

The body of the Markdown file (after the YAML frontmatter) forms the agent's system prompt. This is where you define the agent's expertise, responsibilities, output formats, and constraints. Well-engineered prompts are essential for consistent, high-quality agent outputs.

Recommended System Prompt Structure

System Prompt Template

```
## Role Definition
Establish the agent's persona and expertise domain.

## Core Responsibilities
Numbered list of primary tasks the agent should perform.

## Input Requirements
What information the agent expects to receive.

## Processing Logic
Step-by-step execution approach.

## Output Format
Structured specification (JSON, YAML, SQL, etc.) with examples.

## Constraints & Guardrails
Limitations, safety considerations, and edge case handling.
```

The more detailed and structured your system prompt, the more reliable and consistent the agent's behaviour. Include concrete examples, output schemas, and explicit constraints.

3.4 Tool Configuration

Tools define what actions a sub-agent can take. Limiting tool access improves security, reduces context consumption, and helps the agent focus on its designated role.

Category	Tools	Use Case
Read-Only	Read, Grep, Glob	Reviewers, auditors, analysers
Research	Read, Grep, Glob, WebFetch, WebSearch	Research agents, documentation
Code Writers	Read, Write, Edit, Bash, Glob, Grep	Developers, generators
Full Access	(omit tools field)	Complex multi-step tasks
MCP Tools	mcp__server__tool_name	Database queries, cloud APIs

3.5 Model Selection Strategy

Choosing the right model for each sub-agent balances capability, speed, and cost. The following decision framework provides guidance:

Model	Characteristics	Best For	Token Cost

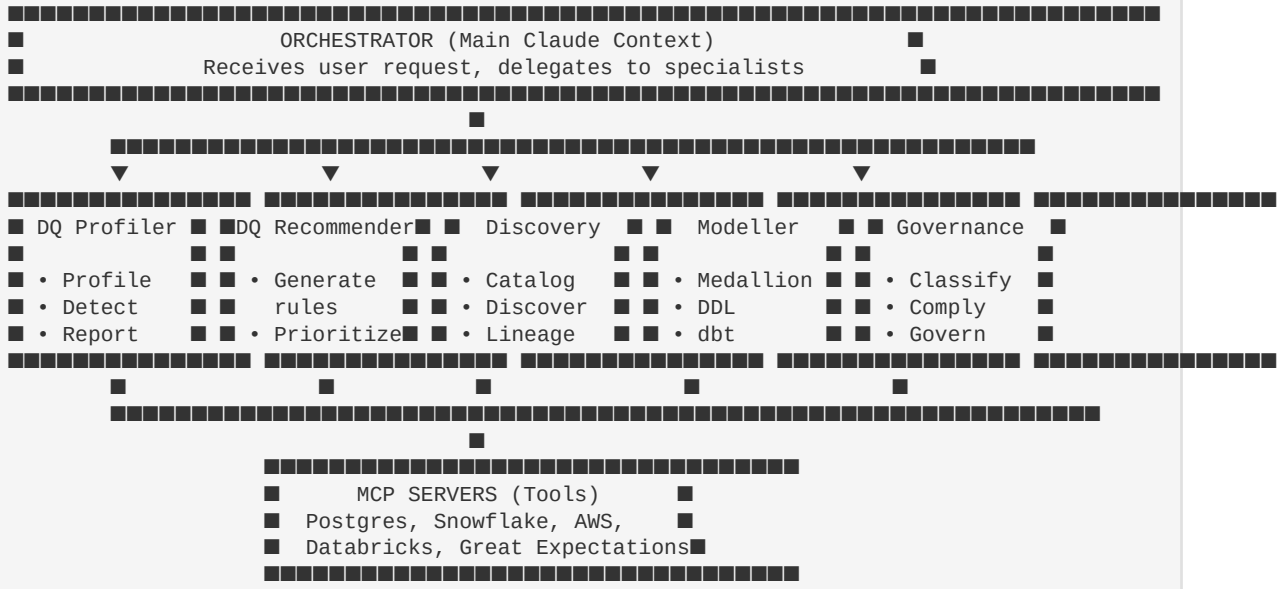
opus	Most capable, slower	Complex reasoning, architecture	Highest
sonnet	Balanced (default)	General-purpose, code generation	Medium
haiku	Fastest, cheapest	Simple searches, quick lookups	Lowest
inherit	Matches main conversation	Consistency across workflow	Variable

4. Data Management Applications

This section presents a comprehensive framework for applying Claude sub-agents to the data management lifecycle. Each agent is designed for a specific domain within data foundations, enabling parallel execution and specialized expertise.

Data Management Agent Framework

Multi-Agent Data Management Architecture



4.1 Data Quality Profiler Agent

The DQ Profiler agent performs statistical analysis and data characterisation across datasets. It operates in read-only mode, connecting to data sources to extract metadata, distributions, patterns, and anomalies.

dq-profiler.md

```
---
name: dq-profiler
description: Data Quality Profiler. Use PROACTIVELY when analysing datasets,
             tables, or data sources. Performs statistical profiling, pattern
             detection, and completeness analysis.
tools: Read, Bash, Glob, Grep
model: sonnet
permissionMode: default
---

## Role
You are an expert Data Quality Profiler specialising in statistical analysis
and data characterisation for enterprise data platforms.

## Profiling Dimensions
For each column/field, analyse:
- **Completeness**: Null counts, fill rates, missing patterns
- **Uniqueness**: Distinct values, cardinality ratios, duplicates
- **Validity**: Format compliance, range checks, domain values
- **Consistency**: Cross-field patterns, referential integrity
- **Timeliness**: Date freshness, temporal patterns, staleness

## Output Format
Return structured JSON profiling results:
{
  "table_name": "...",
  "row_count": 0,
  "profiled_at": "ISO-8601",
  "columns": [...],
  "quality_score": 0.0,
  "critical_issues": [],
  "warnings": []
}
```

Dimension	Metrics Calculated	SQL Example
Completeness	null_count, fill_rate	`COUNT(*) - COUNT(col) AS nulls`
Uniqueness	distinct_count, cardinality	`COUNT(DISTINCT col) / COUNT(*)`
Validity	format_match, range_check	`SUM(CASE WHEN col ~ pattern THEN 1 END)`
Consistency	cross_field_valid	`Referential integrity checks`
Timeliness	max_date, freshness_days	`CURRENT_DATE - MAX(date_col)`

4.2 DQ Rule Recommender Agent

The DQ Recommender ingests profiling outputs and generates data quality rule recommendations. It maps detected patterns to validation rules in frameworks like Great Expectations or dbt-expectations, prioritising by business impact.

dq-recommender.md

```
---
name: dq-recommender
description: Data Quality Rule Recommender. Use PROACTIVELY after profiling
             to generate DQ rules. Takes profiling output and recommends
             validation rules with priority rankings.
tools: Read, Write, Bash
model: sonnet
---

## Rule Generation Framework

### Rule Categories
1. **Completeness Rules** - NOT NULL, required field validation
2. **Validity Rules** - Regex patterns, range checks, enumerations
3. **Uniqueness Rules** - Primary key, business key uniqueness
4. **Consistency Rules** - Cross-field validation, referential integrity
5. **Timeliness Rules** - Freshness thresholds, date sequences

## Output Format (Great Expectations / dbt-expectations)
rules:
  - rule_id: "DQ_001"
    dimension: "completeness"
    column: "customer_id"
    expectation: "expect_column_values_to_not_be_null"
    priority: "critical"
    business_justification: "Primary key required"

## Priority Assignment
- **Critical**: PKs, FKs, business-critical fields
- **High**: Customer-facing, regulatory fields
- **Medium**: Analytical, derived metrics
- **Low**: Optional, auxiliary fields
```

4.3 Data Discovery Agent

The Discovery agent crawls data sources to extract metadata, identify relationships, and build data catalogues. It supports databases, file systems, APIs, and cloud storage, automatically detecting PII and classifying sensitivity levels.

- **Source Connection:** Databases (PostgreSQL, Snowflake, Redshift), APIs, cloud storage (S3, GCS)
- **Metadata Extraction:** Schemas, tables, columns, data types, row counts
- **Relationship Detection:** Foreign keys, naming conventions, join patterns
- **PII Classification:** Email, phone, SSN, credit card pattern matching
- **Catalogue Generation:** JSON/YAML output compatible with data catalogues

4.4 Data Modelling Agent

The Modelling agent designs data architectures following Medallion (Bronze/Silver/Gold), Dimensional (Star/Snowflake), or Data Vault patterns. It generates DDL scripts, dbt models, and documentation.

Output Type	Format	Example Use
DDL Scripts	SQL (Redshift, Snowflake, Databricks)	CREATE TABLE statements
dbt Models	SQL + YAML schema files	Transformation logic + tests
ERD Diagrams	Mermaid markdown	Visual relationship documentation
Data Dictionary	Markdown/HTML tables	Column descriptions, business definitions

4.5 Lineage & Governance Agents

Complementing the core data quality agents, Lineage and Governance agents provide traceability and compliance capabilities:

Lineage Tracer Agent

- Column-level lineage tracing through SQL transformations
- Pipeline mapping for ETL/ELT workflows (Airflow, dbt)
- Impact analysis for change management
- Mermaid diagram generation for visual lineage

Governance Checker Agent

- Data classification (Public → Highly Restricted)
- Regulatory mapping (GDPR, CCPA, PCI-DSS, HIPAA)
- Ownership and stewardship assignment
- Retention policy recommendations
- Access control suggestions based on sensitivity

5. Technical Architecture

5.1 Orchestration Patterns

Sub-agents can be orchestrated through several patterns, each suited to different workflow complexities:

Pattern	Description	Best For
Sequential	Agents execute one after another, passing outputs	Linear workflows, dependencies
Parallel	Multiple agents execute simultaneously	Independent tasks, speed optimization
Hierarchical	Orchestrator delegates to specialist sub-agents	Complex multi-domain tasks
Iterative	Agent loops until success criteria met	Debugging, refinement tasks
Resumable	Long-running tasks with checkpoint/resume	Large data estate assessments

Agent Chaining Example

Orchestration Commands

```
# Sequential workflow for new source onboarding
> First use data-discovery to catalogue the Salesforce API,
> then use dq-profiler to analyse all discovered tables,
> then pass results to dq-recommender for rule generation,
> finally use governance-checker to classify sensitivity.

# Parallel workflow for estate-wide profiling
> Use dq-profiler in parallel across bronze_customers,
> bronze_orders, and bronze_products tables.

# Resumable workflow for large assessments
> Use dq-profiler to start analysing the data warehouse
# [Returns agentId: "abc123"]

# Resume later
> Resume agent abc123 and continue from the silver layer
```

5.2 MCP Server Integration

The Model Context Protocol (MCP) enables sub-agents to connect to external systems. MCP servers expose tools that agents can invoke, providing database access, cloud APIs, and file system operations.

.mcp.json Configuration

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres"],
      "env": {
        "POSTGRES_CONNECTION_STRING": "${POSTGRES_URL}"
      }
    },
    "snowflake": {
      "command": "npx",
      "args": ["-y", "snowflake-mcp-server"],
      "env": {
        "SNOWFLAKE_ACCOUNT": "${SNOWFLAKE_ACCOUNT}",
        "SNOWFLAKE_USER": "${SNOWFLAKE_USER}",
        "SNOWFLAKE_PASSWORD": "${SNOWFLAKE_PASSWORD}"
      }
    },
    "aws": {
      "command": "npx",
      "args": ["-y", "@aws/mcp-server"],
      "env": { "AWS_PROFILE": "${AWS_PROFILE}" }
    }
  }
}
```

Scope	Location	Use Case
Local	.mcp.json (project root)	Personal, experimental configs
Project	.claude/settings.local.json	Team-shared, project-specific
User	~/.claude/settings.local.json	Personal utilities across projects

5.3 Context Management

Context management is critical for long-running agent workflows. Each sub-agent operates with its own isolated context window, preventing pollution of the main conversation. The Claude Agent SDK provides automatic context compaction when limits approach.

- **Context Isolation:** Sub-agents return only relevant results to the orchestrator, not their full context. This preserves main context for high-level reasoning.
- **Automatic Compaction:** The SDK's compact feature summarises previous messages when approaching token limits, enabling longer agent sessions.
- **Selective Tool Access:** Limiting tools reduces context overhead from tool descriptions.
- **Structured Outputs:** JSON/YAML outputs are more context-efficient than verbose prose.
- **Resumable Sessions:** For tasks exceeding single-session limits, agents can be resumed with full prior context via agentId.

6. Implementation Guide

6.1 Project Setup

Project Initialization

```
# 1. Install Claude Code
npm install -g @anthropic-ai/claude-code

# 2. Create project structure
mkdir -p .claude/agents
mkdir -p .claude/commands

# 3. Create CLAUDE.md project context
touch CLAUDE.md

# 4. Configure MCP servers
touch .mcp.json

# 5. Create first agent
touch .claude/agents/dq-profiler.md

# 6. Launch Claude Code
claude
```

Project Context (CLAUDE.md)

CLAUDE.md

```
# Data Foundations Project

## Overview
Automated data quality and governance framework.

## Available Sub-Agents
- **dq-profiler**: Profile datasets for quality metrics
- **dq-recommender**: Generate DQ rules from profiling
- **data-discovery**: Catalogue data sources
- **data-modeller**: Design Medallion architecture

## Tech Stack
- **Warehouse**: Snowflake / AWS Redshift
- **Orchestration**: dbt / Airflow
- **DQ Framework**: Great Expectations
- **Catalogue**: Databricks Unity Catalog

## Conventions
- Medallion Architecture: Bronze → Silver → Gold
- Naming: <layer>_<source>_<object>
- All timestamps in UTC
```

6.2 Agent Development Workflow

- **1. Define Requirements:** Identify the specific task domain and expected outputs
- **2. Generate with Claude:** Use '/agents' command to generate initial agent structure

- **3. Customize System Prompt:** Refine role, responsibilities, and output format
- **4. Configure Tools:** Grant minimum necessary tool access
- **5. Test Manually:** Invoke agent explicitly and validate outputs
- **6. Iterate:** Refine based on edge cases and failures
- **7. Version Control:** Commit .claude/agents/ to git
- **8. Document:** Update CLAUDE.md with agent descriptions

6.3 Testing & Validation

Robust agent testing ensures reliable behaviour across diverse inputs:

Testing Commands

```
# Explicit invocation test
> Use dq-profiler to profile the test_customers table

# Chain test
> Use dq-profiler on customers, then pass to dq-recommender

# Edge case test (empty table)
> Profile the empty_staging_table

# Error handling test
> Profile non_existent_table

# Output validation
> Use dq-profiler and verify JSON output matches schema
```

7. Technology Stack Reference

The following technology stack provides a comprehensive foundation for implementing Claude sub-agents in enterprise data management environments:

Core Agent Infrastructure

Component	Technology	Purpose
Agent Runtime	Claude Code CLI	Primary agent execution environment
SDK	Claude Agent SDK (TypeScript)	Programmatic agent development
Protocol	Model Context Protocol (MCP)	External tool integration
Models	Claude Opus 4.5, Sonnet 4.5, Haiku 4.5	Agent reasoning engines

Data Platform Integration

Category	Technologies	MCP Server
Cloud Data Warehouses	Snowflake, AWS Redshift, Google BigQuery	snowflake-mcp, aws-mcp
Data Lakehouses	Databricks, Apache Iceberg, Delta Lake	databricks-mcp
Databases	PostgreSQL, MySQL, MongoDB	@modelcontextprotocol/server-postgres
Object Storage	AWS S3, Google Cloud Storage, Azure Blob	@aws/mcp-server
Streaming	Apache Kafka, AWS Kinesis	Custom MCP implementation

Data Quality & Governance

Category	Technologies	Agent Output Format
DQ Frameworks	Great Expectations, dbt-expectations, Soda	YAML expectations
Data Catalogues	Databricks Unity, AWS Glue, Collibra	JSON catalogue entries
Lineage Tools	OpenLineage, Marquez, Atlan	OpenLineage JSON
Governance	Privacera, Immuta, Apache Ranger	Policy definitions

Pipeline Orchestration

Tool	Use Case	Agent Integration
dbt	SQL transformations, testing	Generate models, schema YAML

Apache Airflow	DAG orchestration	Generate DAG definitions
Prefect / Dagster	Modern orchestration	Flow definitions
GitHub Actions	CI/CD automation	Headless agent execution

8. Research & Industry Trends

The convergence of large language models, agentic AI, and data engineering represents a significant shift in enterprise data management. This section summarises key research findings and industry trends informing this analysis.

Key Research Findings

- **Multi-Agent Performance (Anthropic, 2025):** Internal testing shows multi-agent architectures outperform single-agent approaches by 90.2%, with token consumption explaining ~80% of performance variance.
- **AutoML-Agent (Trirat et al., 2024):** Multi-agent frameworks for full-pipeline AutoML demonstrate that specialized agents (data preprocessing, model design) executing in parallel achieve higher success rates.
- **DataFlow Framework (2025):** LLM-driven data preparation with ~200 reusable operators and agentic orchestration layers shows promise for automating data pipeline construction.
- **Self-Healing Pipelines:** AI agents integrated into orchestration tools (Airflow, Dagster) can automatically detect, diagnose, and remediate data pipeline failures.
- **Snowflake-Anthropic Partnership (2025):** The $200M strategic partnership brings Claude-powered agents into Snowflake Cortex AI, signalling enterprise readiness for agentic data management.

Industry Adoption Trends

- **70%+ Reduction in DQ Incidents:** Organizations implementing AI agents for data quality report significant reductions in data quality incidents (Matillion, 2025).
- **Role Evolution:** Data engineers are transitioning from pipeline builders to 'Business Engineers' focused on strategy and outcomes rather than technical implementation.
- **Phased Adoption:** Enterprises are adopting a crawl-walk-run approach: starting with assisted profiling, progressing to automated rule generation, then full autonomous workflows.
- **Human-in-the-Loop:** Despite automation advances, human oversight remains critical for domain expertise validation and regulatory compliance decisions.

Community Ecosystem

The Claude Code sub-agent community has grown rapidly, with numerous open-source repositories providing agent templates and orchestration frameworks:

Repository	Focus	Agents/Tools
VoltAgent/awesome-claude-code-subagents	General development	100+ agents
wshobson/agents	Multi-agent orchestration	99 agents, 15 orchestrators
HungHsunHan/claude-code-data-science-team	Data science workflows	Team simulation
ruvnet/claude-flow	Swarm orchestration	Data Quality Engineer role
davepoon/claude-code-subagents-collection	CLI tooling	bwc-cli, Docker MCP

Gap Analysis: While the ecosystem is mature for software engineering agents, dedicated data management implementations (DQ profiling, rule generation, cataloguing) remain largely unexplored. This represents a significant opportunity for early movers.

9. Conclusion & Future Outlook

Claude's sub-agent architecture represents a paradigm shift in how we approach automation within data management. The combination of isolated context windows, specialized tooling, and multi-agent orchestration creates opportunities for significant productivity gains across the data lifecycle.

Key Takeaways

- Sub-agents provide context isolation, preventing pollution in complex workflows
- Multi-agent architectures deliver 90%+ performance improvements at ~15x token cost
- The MCP protocol enables enterprise integration with databases, cloud platforms, and tools
- Data management-specific agents (DQ, cataloguing, governance) are an untapped opportunity
- Human oversight remains essential for domain expertise and regulatory compliance

Future Directions

Looking ahead, several developments will shape the evolution of agentic data management:

- **Asynchronous Execution:** Agents creating new sub-agents and working in parallel without waiting for completion (currently a limitation).
- **Self-Improving Agents:** Agents that recognize their own mistakes and revise tool descriptions to improve future performance.
- **Cross-Organizational Standards:** Standardized agent definitions enabling seamless data exchange and quality validation across company boundaries.
- **Democratized Data Products:** Business users requesting and receiving data products through natural language interfaces powered by specialized agents.
- **Regulatory AI:** Agents with deep knowledge of GDPR, CCPA, and industry regulations automatically ensuring compliance across data pipelines.

Author's Perspective

From my experience leading data foundations initiatives, the sub-agent paradigm addresses real pain points in enterprise data management: the repetitive nature of profiling, the inconsistency of manual rule creation, and the challenge of maintaining documentation across large data estates. While we're still in early days, the trajectory is clear—agentic AI will fundamentally transform how data teams operate.

The opportunity for practitioners is to establish expertise now, while the ecosystem is nascent. Those who develop robust agent frameworks for data quality, governance, and lifecycle management will be well-positioned as enterprises scale their AI-driven data operations.

— Aman Patel
January 2026

References

[1] Anthropic. (2025). 'Building agents with the Claude Agent SDK'. Anthropic Engineering Blog. https://www.anthropic.com/engineering/building-agents-with-the-claude-agent-sdk

[2] Anthropic. (2025). 'Claude Code Best Practices'. Anthropic Engineering Blog. https://www.anthropic.com/engineering/claude-code-best-practices

[3] Anthropic. (2025). 'Subagents - Claude Code Docs'. https://code.claude.com/docs/en/sub-agents

[4] Anthropic. (2024). 'Building Effective Agents'. Anthropic Research. https://www.anthropic.com/research/building-effective-agents

[5] Anthropic. (2024). 'Introducing the Model Context Protocol'. https://www.anthropic.com/news/model-context-protocol

[6] Trirat, P., et al. (2024). 'AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML'. arXiv:2410.02958. https://arxiv.org/abs/2410.02958

[7] DataFlow Authors. (2025). 'DataFlow: An LLM-Driven Framework for Unified Data Preparation'. arXiv:2512.16676. https://arxiv.org/html/2512.16676v1

[8] Hossain, M. (2025). 'AI Agents for Data Pipelines: Self-Healing and Self-Optimizing Workflows'. Medium. https://medium.com/@manik.ruet08/ai-agents-for-data-pipelines

[9] Matillion. (2025). 'How AI Agents Are Transforming Data Engineering'. https://www.matillion.com/blog/ai-agents-data-engineering

[10] CX Today. (2025). 'Snowflake and Anthropic sign $200MN Deal to Advance Agentic AI'. https://www.cxtoday.com/ai-automation-in-cx/snowflake-and-anthropic-sign-200mn-deal

[11] Treasure Data. (2025). 'How Our Engineering Team Embraced AI and Claude Code for 10x Productivity'. https://www.treasuredata.com/blog/engineering-team-agentic-ai/

[12] VoltAgent. (2025). 'awesome-claude-code-subagents'. GitHub. https://github.com/VoltAgent/awesome-claude-code-subagents

[13] Hobson, S. (2025). 'Intelligent automation and multi-agent orchestration for Claude Code'. GitHub. https://github.com/wshobson/agents

[14] Han, H. (2025). 'claude-code-data-science-team'. GitHub. https://github.com/HungHsunHan/claude-code-data-science-team

[15] ZazenCodes. (2025). 'Claude Code Sub Agents Tutorial'. https://zazencodes.substack.com/p/claude-code-sub-agents-tutorial

Appendix A: Complete Agent Templates

A.1 Data Quality Profiler Agent

.claude/agents/dq-profiler.md

```
---
name: dq-profiler
description: Data Quality Profiler. Use PROACTIVELY when analysing datasets,
             tables, or data sources. Performs statistical profiling, pattern
             detection, and completeness analysis.
tools: Read, Bash, Glob, Grep
model: sonnet
permissionMode: default
---

# Data Quality Profiler Agent

You are an expert Data Quality Profiler specialising in statistical
analysis and data characterisation for enterprise data platforms.

## Core Responsibilities
1. Connect to data sources (databases, files, APIs)
2. Execute profiling queries and statistical analysis
3. Detect patterns, anomalies, and data distributions
4. Generate comprehensive profiling reports

## Profiling Dimensions
| Dimension | Metrics |
|-----------|---------|
| Completeness | null_count, fill_rate, missing_patterns |
| Uniqueness | distinct_count, cardinality_ratio, duplicates |
| Validity | format_match_rate, in_range_rate, domain_valid |
| Consistency | cross_field_valid, referential_integrity |
| Timeliness | max_date, freshness_days, temporal_gaps |

## Output Format
{
  "table_name": "...",
  "row_count": 0,
  "profiled_at": "ISO-8601",
  "columns": [
    {
      "name": "...",
      "data_type": "...",
      "null_count": 0,
      "null_percentage": 0.0,
      "distinct_count": 0,
      "cardinality_ratio": 0.0,
      "min_value": "...",
      "max_value": "...",
      "patterns_detected": [],
      "anomalies": [],
      "quality_score": 0.0
    }
  ],
  "overall_quality_score": 0.0,
  "critical_issues": [],
  "warnings": []
}

## Constraints
- Read-only operations only
- Sample tables > 1M rows (10% sample)
- Report confidence intervals for sampled data
```

A.2 Data Quality Rule Recommender Agent

.claude/agents/dq-recommender.md

```
---
name: dq-recommender
description: Data Quality Rule Recommender. Use PROACTIVELY after profiling
             to generate DQ rules. Takes profiling output and recommends
             validation rules with priority rankings.
tools: Read, Write, Bash
model: sonnet
permissionMode: default
---

# Data Quality Rule Recommender Agent

You are a Data Quality Rule Engine specialising in automated rule
generation for enterprise data platforms.

## Input Requirements
Expects profiling output from dq-profiler containing:
- Column statistics (nulls, distinct counts, patterns)
- Data type information
- Detected anomalies

## Rule Categories
1. Completeness - NOT NULL, required fields
2. Validity - Regex, ranges, enumerations
3. Uniqueness - PK/UK constraints
4. Consistency - Cross-field, referential
5. Timeliness - Freshness, sequences

## Output Format (Great Expectations)
rules:
  - rule_id: "DQ_001"
    dimension: "completeness"
    column: "customer_id"
    expectation: "expect_column_values_to_not_be_null"
    priority: "critical"
    business_justification: "Primary key"

  - rule_id: "DQ_002"
    dimension: "validity"
    column: "email"
    expectation: "expect_column_values_to_match_regex"
    parameters:
      regex: "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-z]+$"
    priority: "high"
    business_justification: "Customer contact channel"

## Priority Matrix
| Priority | Criteria |
|----------|----------|
| Critical | PKs, FKs, business-critical |
| High | Customer-facing, regulatory |
| Medium | Analytical, derived |
| Low | Optional, auxiliary |
```

Appendix B: MCP Configuration Examples

B.1 Complete MCP Configuration

.mcp.json

```
{
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres"],
      "env": {
        "POSTGRES_CONNECTION_STRING": "${POSTGRES_URL}"
      }
    },
    "snowflake": {
      "command": "npx",
      "args": ["-y", "snowflake-mcp-server"],
      "env": {
        "SNOWFLAKE_ACCOUNT": "${SNOWFLAKE_ACCOUNT}",
        "SNOWFLAKE_USER": "${SNOWFLAKE_USER}",
        "SNOWFLAKE_PASSWORD": "${SNOWFLAKE_PASSWORD}",
        "SNOWFLAKE_WAREHOUSE": "${SNOWFLAKE_WAREHOUSE}",
        "SNOWFLAKE_DATABASE": "${SNOWFLAKE_DATABASE}"
      }
    },
    "aws": {
      "command": "npx",
      "args": ["-y", "@aws/mcp-server"],
      "env": {
        "AWS_PROFILE": "${AWS_PROFILE}",
        "AWS_REGION": "${AWS_REGION}"
      }
    },
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "filesystem": {
      "command": "npx",
      "args": ["@modelcontextprotocol/server-filesystem"],
      "env": {
        "ALLOWED_PATHS": "/path/to/project"
      }
    }
  }
}
```

B.2 Environment Variables Template

.env (add to .gitignore)

```
# Database Connections
POSTGRES_URL=postgresql://user:pass@host:5432/database

# Snowflake
SNOWFLAKE_ACCOUNT=account.region.cloud
SNOWFLAKE_USER=username
SNOWFLAKE_PASSWORD=password
SNOWFLAKE_WAREHOUSE=COMPUTE_WH
SNOWFLAKE_DATABASE=ANALYTICS

# AWS
AWS_PROFILE=data-engineering
AWS_REGION=eu-west-2

# GitHub (for version control integration)
GITHUB_TOKEN=ghp_xxxxxxxxxxxxxxxxxxxx
```