Architectural Design and Benchmarking Framework for Agentic Salesforce Development: A Comprehensive Technical Report

1. Executive Summary

The emergence of agentic Artificial Intelligence (AI) necessitates a paradigm shift in how we evaluate model capabilities. While current benchmarks like SWE-bench effectively assess code generation in file-based languages such as Python or Java, they fail to capture the architectural complexity of Platform-as-a-Service (PaaS) environments like Salesforce. Salesforce development is not merely a coding discipline; it is a hybrid practice requiring the orchestration of declarative metadata, proprietary programming languages (Apex, SOQL), and stateful database interactions within a multi-tenant environment governed by strict execution limits.

This report presents a rigorous technical design for **SF-AgentBench**, a specialized benchmarking framework designed to evaluate AI agents—powered by tools like Claude Code, Codex, or Gemini Orchestrator—on their ability to design and build Salesforce solutions. The framework is grounded in the official Salesforce curriculum, specifically mapping to the Administrator (ADM-201) and Platform Developer (PD1/PD2) certifications. It utilizes the "Superbadge" methodology—complex, scenario-based problem solving—as the gold standard for evaluation, moving beyond atomic code generation to holistic solution architecture.

The proposed architecture introduces a novel **Agent-Computer Interface (ACI)** that wraps the Salesforce CLI (sf), enabling agents to operate securely within ephemeral Scratch Orgs. This design addresses the unique challenges of the domain: the coupling of metadata and data, the enforcement of Governor Limits, and the necessity of destructive testing. By integrating multi-layered evaluation metrics—including deployment validation, functional Apex testing, static code analysis (PMD), and metadata configuration diffing—the framework provides a nuanced assessment of an agent's true capability. This report details the skills taxonomy, the orchestration architecture, the design of agentic tools, and the scoring methodologies required to distinguish between hallucinated configuration and functional enterprise delivery.

2. Domain Analysis: The Salesforce Competency

Taxonomy

To construct a valid benchmark, one must first deconstruct the domain into testable atomic and composite skills. The Salesforce ecosystem presents a unique challenge for AI models trained on general-purpose codebases because the "source of truth" is often split between the local file system (metadata) and the cloud environment (database schema and configuration).

2.1 The Hybrid Nature of Salesforce Engineering

Salesforce engineering is characterized by the coexistence of **Declarative Development** (low-code/no-code) and **Programmatic Development** (code). An effective AI agent must understand when to apply which paradigm and how they interact. This hybrid nature creates specific failure modes for AI agents that pure coding benchmarks miss.

- **Metadata Dependency:** A programmatic trigger often relies on a declarative field existing on an object. An agent failing to deploy the CustomField metadata before the ApexTrigger will encounter a deployment failure. This requires the agent to understand dependency graphs and deployment order, a skill distinct from syntax generation.[1]
- **Governor Limits:** Unlike local execution environments (like a Python container), Salesforce enforces strict runtime limits (SOQL queries, CPU time, heap size) to ensure multi-tenant stability. Agents must generate "bulkified" code that processes collections of records rather than iterating individual operations. A structurally correct Python function might translate to a disastrously inefficient Apex trigger if the agent does not conceptually grasp "bulkification".[3]
- **Contextual Security:** Visibility of data is controlled by a complex intersection of Profiles, Permission Sets, and Sharing Rules. An agent might write a perfect query, but if the running user lacks the Permission Set to see the field, the query returns null. Agents must demonstrate the ability to configure these security layers correctly to ensure the code actually functions in production.[5]

2.2 Administrator Skills Curriculum (ADM-201 & Advanced Admin)

The benchmark must test the following administrative domains, derived from the ADM-201 exam guide and real-world administrative tasks.[1]

2.2.1 Configuration and Object Architecture

- **Schema Management:** Creating Custom Objects, Fields (Master-Detail, Lookup, Formula), and Page Layouts. The agent must understand the implications of field types on reporting and relationships. For instance, converting a Master-Detail relationship to a Lookup has profound security implications regarding record ownership that an agent must navigate.
- **Data Validation:** Implementing Validation Rules using formulas (e.g., AND, OR, ISBLANK, VLOOKUP). This tests the agent's logical reasoning and understanding of Salesforce

formula syntax, which differs significantly from Excel or standard programming logic. The agent must construct boolean logic that evaluates to TRUE to *block* an action, a "negative logic" pattern that often confuses basic models.[8]
- **User Interface:** Configuring Lightning App Builder pages, Record Types, and assigning them to Profiles. This involves manipulating XML metadata for FlexiPage and Layout components.

2.2.2 Process Automation

- **Flow Orchestration:** Designing Screen Flows, Record-Triggered Flows, and Autolaunched Flows. As Salesforce retires Workflow Rules and Process Builder, Flow has become the primary automation engine. The benchmark must test the agent's ability to generate the complex XML metadata for a Flow, which includes <decisions>, <loops>, <assignments>, and <recordCreates>. This is effectively visual programming serialized into XML.[10]
- **Approval Processes:** Defining multi-step approval logic, entry criteria, and approver assignment. The agent must configure the ApprovalProcess metadata, defining steps, rejection actions, and recall actions.[12]

2.2.3 Security and Access

- **Security Model:** Configuring Organization-Wide Defaults (OWD), Role Hierarchies, and Sharing Rules.
- **Permissions:** Creating Permission Sets and Profiles to grant granular access to objects and fields. The agent must be able to modify the Profile metadata to set <fieldPermissions> and <objectPermissions> correctly without overwriting existing settings.[5]

2.3 Developer Skills Curriculum (PD1 & PD2)

The developer track requires the agent to manipulate the platform via code, tested against the PD1 and PD2 competencies.[2]

2.3.1 Apex Fundamentals and Logic

- **Triggers:** Writing triggers that handle events (before insert, after update) and adhering to the "One Trigger Per Object" pattern. The agent must handle Trigger.new, Trigger.old, and context variables (Trigger.isInsert, Trigger.isBefore). It must demonstrate recursion handling to prevent stack depth overflows.[3]
- **Object-Oriented Design:** Implementing interfaces, abstract classes, and virtual methods within the Apex type system.
- **Testing:** Writing IsTest classes with System.assert to validate logic and ensuring 75% code coverage. This includes using Test.startTest() and Test.stopTest() to reset governor limits and testing user contexts with System.runAs().[17]

2.3.2 Data Integration and Asynchronous Processing

- **SOQL/SOSL:** Constructing efficient queries, understanding relationship queries (Parent-to-Child subqueries, Child-to-Parent dot notation), and strictly avoiding queries inside loops.
- **Asynchronous Apex:** Implementing Batchable, Queueable, Schedulable, and Future methods for long-running processes. The agent must understand when to use Queueable (chaining jobs) vs Future (simple primitive parameter calls).[14]
- **APIs:** Designing Apex REST services (@RestResource) and consuming external web services (WSDL to Apex, HTTP Callouts). The agent must handle authentication via Named Credentials rather than hardcoding endpoints.[20]

2.3.3 User Interface Development

- **Lightning Web Components (LWC):** Writing JavaScript controllers, HTML templates, and CSS. The agent must understand the reactive wire service (@wire) for data provisioning and the interaction with Apex controllers via @AuraEnabled methods. It must also handle the folder structure requirements of LWC bundles (js, html, js-meta.xml).[21]

3. Benchmarking Framework Design: SF-AgentBench

The **SF-AgentBench** framework is designed to bridge the gap between abstract coding capabilities and platform-specific execution. Unlike a local Python benchmark, a Salesforce benchmark must manage a cloud environment. The proposed architecture adopts a tiered approach that moves from simple atomic tasks to complex "Superbadge-level" scenarios.

3.1 Architecture Overview

The benchmark operates as a middleware between the AI Agent and the Salesforce Platform. It manages the lifecycle of the test environment (Scratch Org), provides the agent with necessary context, and evaluates the resulting state using a combination of deployment validation, unit testing, and metadata analysis.

Core Components:

1. **Task Registry:** A database of tasks derived from Trailhead modules and Superbadges. Each task contains the Prompt, the "Golden" Metadata (solution), Verification Scripts (Apex tests/Node scripts), and the project-scratch-def.json required to spin up the correct environment.
2. **Orchestration Engine:** A Python/Node.js application that manages the queue of evaluations. It spins up Docker containers for individual agent runs to ensure isolation.[23]
3. **Salesforce Environment Layer:** Utilizes **Scratch Orgs** created via a Dev Hub. These provide a clean, disposable state for every test run, ensuring no interference between benchmark iterations.[24]

4. **Evaluation Layer:** A multi-modal scoring system combining sf CLI outputs, Apex Test results, Metadata Diffing (using sfdx-git-delta), and Static Analysis (Salesforce Code Analyzer).

3.2 Environment Management: The Scratch Org Lifecycle

A critical requirement for valid benchmarking is environment isolation. SF-AgentBench utilizes **Scratch Orgs** defined by project-scratch-def.json files to create deterministic starting states. This is superior to using a persistent Sandbox, which suffers from configuration drift and data pollution.

Lifecycle Automation:

1. **Initialization:** The benchmark reads the task configuration and selects the appropriate project-scratch-def.json (e.g., enabling "Communities" or "MultiCurrency"). It executes sf org create scratch to provision the environment.[25]
2. **Dependencies:** If the task requires pre-installed packages (e.g., the "Apex Specialist" unmanaged package 04t...), the orchestrator installs them using sf package install. This mimics a real-world scenario where developers build on top of existing managed packages.[27]
3. **Data Seeding:** An empty org is useless for testing logic. Sample data is loaded using sf data tree import or sfdx-data-move-utility. This ensures the agent has records to test against (e.g., Accounts, Contacts, Custom Objects).[28]
4. **Execution:** The agent is given access to the local DX project directory and authentication aliases to perform its work. It interacts with the org via the provided toolset.
5. **Teardown:** Upon completion or timeout, sf org delete scratch is invoked to free up allocations. Given the limits on Scratch Org creation (e.g., 6 per day for Developer Edition, significantly more for Partners), the architecture must support **Org Pooling**—pre-creating orgs and keeping them "warm" to reduce the latency of the benchmark run.[30]

3.3 Data Strategy: The Foundation of Verification

Salesforce logic is data-dependent. A trigger that updates an Account when a Contact is saved can only be verified if both Account and Contact records exist.

- **Tree Import:** For hierarchical data (Parent-Child), the sf data import tree command is essential. It maintains relationship integrity by resolving foreign keys during import. The benchmark must provide pre-built JSON plans (plan.json) for data seeding.[28]
- **Synthetic Data Generation:** For stress testing (bulkification), the framework should include scripts to generate large volumes of data (e.g., 200+ records) to verify that the agent's code does not hit Governor Limits.

3.4 Task Design: From Superbadges to Benchmarks

Superbadges (e.g., Apex Specialist, Process Automation Specialist) are ideal candidates for high-level benchmarks because they require synthesizing multiple skills to solve a business problem.[33]

Example Task Construction (Apex Specialist Challenge 1):

- **Prompt:** "Automate the maintenance request routine. When a Case of type 'Repair' or 'Routine Maintenance' is closed, automatically create a new Routine Maintenance Case for the same Vehicle and Equipment. The new due date should be calculated based on the equipment's maintenance cycle."
- **Starting State:** A standard DX project structure with Case, Vehicle__c, and Equipment__c objects.
- **Agent Tools:** sf apex generate class, sf project deploy start, File System Access.
- **Success Criteria:**
  - Existence of MaintenanceRequest trigger.
  - Existence of MaintenanceRequestHelper class (Best Practice separation).
  - Passing MaintenanceRequestHelperTest with 100% coverage.
  - Functional correctness verified by a hidden test class that inserts a closed Case and queries for the new Case.[35]

4. Technical Design for Agentic Tool Creation

To enable agents (Claude Code, Codex, Gemini) to interact with Salesforce effectively, we must define an **Agent-Computer Interface (ACI)** specific to the sf CLI and Salesforce metadata structure. This aligns with the methodologies used in SWE-agent to simplify the shell interface for LLMs, reducing token usage and focusing the model on high-level reasoning.[23]

4.1 The Salesforce CLI Tool Wrapper

Raw CLI output can be verbose and unstructured. The agentic tool wrapper should normalize inputs and outputs to reduce token usage and improve reasoning. The tools should be exposed via an API that the agent can call (e.g., function calling).

Tool Definitions:

1. **search_metadata**:
   - **Purpose:** Allows the agent to find existing metadata components without dumping the entire file tree.
   - **Implementation:** Wraps sf project list metadata or searches the local force-app directory using grep or find.
   - **Agent Input:** { type: "ApexClass", name: "*Controller" }
   - **Rationale:** Salesforce projects can have thousands of files. Agents need a way to

navigate efficiently.

2. **retrieve_org_state**:
    - **Purpose:** Syncs the local directory with the Org's state.
    - **Implementation:** Executes sf project retrieve start.
    - **Context:** Essential for Admins who might make changes in the UI (if the agent simulates a browser) or to validate that a deployment actually succeeded and reflected in the local source.
3. **deploy_metadata**:
    - **Purpose:** Pushes changes to the scratch org.
    - **Implementation:** Wraps sf project deploy start --json.
    - **Output Processing:** The wrapper parses the JSON output to provide a concise summary: "Deployment Failed: Field 'Status__c' not found on Object 'Case'." This direct feedback loop is critical for the agent's iterative correction. Raw CLI output is often too noisy for effective LLM parsing.[38]
4. **run_apex_tests**:
    - **Purpose:** Validates logic and coverage.
    - **Implementation:** Wraps sf apex run test --code-coverage --result-format human.
    - **Optimization:** The tool should allow running specific test classes (--class-names) to save time, rather than RunAllTests every time. It should return specific failure messages and stack traces.[17]
5. **query_data**:
    - **Purpose:** Allows the agent to inspect record data to verify automation logic (e.g., "Did the Flow create the Task?").
    - **Implementation:** Wraps sf data query --query "SELECT Id, Subject FROM Task".
    - **Insight:** This tool allows the agent to act as its own QA engineer, verifying the "State" of the database after an action.
6. **describe_object**:
    - **Purpose:** Provides schema details.
    - **Implementation:** Wraps sf sobject describe --sobject <Name>.
    - **Output:** Returns field names, types, and relationships. This prevents hallucination of field API names (e.g., guessing Email_Address__c when it is Email__c).

4.2 Handling XML and Metadata Verbosity

Salesforce metadata (XML) is highly verbose. An LLM context window can be quickly consumed by a single Profile or CustomObject file (which can be thousands of lines).

Optimization Strategy:

- **Fragmented Views:** The tool should allow reading specific sections of metadata. For example, instead of reading the entire Account.object-meta.xml, the agent could request read_field_definition(object="Account", field="Industry").
- **Skeleton Generation:** When creating new metadata (e.g., a new FlexiPage), the tool can provide a skeleton XML to the agent, requiring it only to fill in the relevant component

properties. This reduces the likelihood of hallucinated, invalid XML tags that break the deployment parser.

4.3 Agent Sandbox Isolation

The agent code must run in a secure, sandboxed environment to prevent unauthorized access to the host machine or other Salesforce orgs.

- **Containerization:** Each benchmark run occurs in a Docker container equipped with Node.js, Java (for Data Loader/CLI), and the sf CLI.[23]
- **Network Policies:** The container is restricted to communicating only with the Salesforce login endpoints (login.salesforce.com, test.salesforce.com) and the specific Scratch Org instance.
- **Credential Injection:** Authentication tokens (JWT or Auth URL) are injected as environment variables at runtime and stripped upon container destruction. The sf CLI authentication is handled via sf org login jwt or sf org login sfdx-url using these injected secrets.[40]

5. Evaluation Methodology: Scoring and Metrics

Evaluating Salesforce solutions requires a multi-dimensional approach. A simple "Pass/Fail" is insufficient for grading complex Admin or Developer tasks where multiple valid solutions may exist, or where a solution may be partially correct (e.g., logic is correct, but security settings are missing).

5.1 The Tiered Scoring Pyramid

Level 1: Deployment Validation (Syntactic Correctness)

The most basic metric is whether the metadata deploys to the Org without error.

- **Tool:** sf project deploy validate or sf project deploy start --checkonly.
- **Metric:** Deployment Success Rate. If the agent generates XML that violates the schema (e.g., invalid tag in a Flow, circular dependency), it receives a score of 0 for this tier. This measures the agent's syntax proficiency.

Level 2: Functional Verification (Apex Testing)

For both programmatic and declarative automation (Flows), functional correctness is verified via Apex tests.

- **Mechanism:** The benchmark injects a hidden "Validator" test class into the deployment that the agent cannot see or modify. This class creates data, triggers the automation, and asserts the results.
- **Flow Testing:** Since Flows create records, Apex tests can assert that the expected records exist. For example, "Insert an Opportunity -> Assert a Task was created." This bridges the gap between Admin and Dev testing.[41]

- **Metric:** Test Pass Rate (percentage of assertions passed).

Level 3: Metadata Compliance (Static Analysis)

Does the solution follow best practices?

- **Tool:** Salesforce Code Analyzer (sf scanner run) using PMD and ESLint engines.[43]
- **Checks:**
  - **Security:** Detecting SOQL injection vulnerabilities.
  - **Performance:** Detecting SOQL queries inside loops (a critical Salesforce anti-pattern).
  - **Standards:** Naming conventions, hardcoded IDs.
- **Metric:** Quality Score (100 - penalty points for violations). This encourages agents to write maintainable, enterprise-grade code.

Level 4: Configuration Integrity (Metadata Diff)

For Admin tasks (e.g., "Set OWD to Private"), Apex tests are sometimes insufficient as they test behavior, not configuration. We must verify the configuration state directly.

- **Tool:** sfdx-git-delta or sf project retrieve.
- **Methodology:** The benchmark retrieves the specific metadata (e.g., Account.object) and compares specific XML nodes against a "Golden" solution file using a JSON diff approach.
- **Example:** Checking if <sharingModel>Private</sharingModel> exists in the CustomObject metadata.
- **Metric:** Configuration Accuracy (percentage of required XML nodes present and correct).

5.2 Partial Credit and Rubrics

Salesforce tasks are rarely binary. An agent might correctly write a Trigger but fail to bulkify it.

- **Rubric-Based Scoring:** Utilizing an LLM-as-a-judge to evaluate the generated code/metadata against a rubric.[45]
- **Rubric Example (Trigger Task):**
  - **Logic Correctness (40%):** Does it update the child record? (Verified by Test)
  - **Bulkification (30%):** Does it use Maps/Lists instead of iterating queries? (Verified by Static Analysis/PMD)
  - **Best Practice (20%):** Is logic in a handler class, not the trigger body? (Verified by File Structure Check)
  - **Coverage (10%):** Is there a test class? (Verified by Coverage Report)

5.3 Benchmarking Framework Comparison

Why build a new benchmark instead of using existing ones?

- **SWE-bench:** Excellent for Python/Java but lacks the Salesforce context (metadata, governor limits, cloud state). It cannot evaluate if a Flow is correct or if FLS is set.
- **HumanEval:** Too atomic. Salesforce development is about *integration* of components, not just writing a sorting algorithm.
- **OSWorld:** Good for GUI interaction, but Salesforce development is increasingly moving to CLI/Code (DX). OSWorld is too heavy for evaluating metadata generation.
- **SF-AgentBench:** Fills this gap by treating the Salesforce Org as the "Environment" and the Metadata as the "Code," bridging the two via the CLI.

6. Implementation Strategy: Skills to Benchmark

Based on the research, the following specific skills should be prioritized for the benchmark, categorized by their complexity and the testing framework required.

6.1 Administrator Benchmark Tasks

Skill Area	Task Example	Verification Method	Complexity
Object Manager	"Create a 'Projects' object with a Master-Detail relationship to Account."	Metadata Diff (Check CustomObject XML for <type>MasterDetail</type>).	Low
Validation Rules	"Prevent closed opportunities from being edited unless by an Admin."	Functional Test (Run Apex test as Standard User try edit -> assert error).	Medium
Flow Automation	"Create a Record-Triggered Flow that posts to Chatter when a Lead is converted."	Functional Test (Convert Lead -> Query FeedItem table).	High
Security	"Create a Permission Set 'Recruiter' granting	Metadata Diff (Check PermissionSet XML	Medium

	Read/Create on Candidates."	for objectPermissions).	
Data Import	"Import 50 Leads from a CSV file ensuring no duplicates."	Query Data (Count records) + Duplicate Rule verification.	Medium

6.2 Developer Benchmark Tasks

Skill Area	Task Example	Verification Method	Complexity
Apex Triggers	"Write a trigger on Contact to update the Account's 'Active Contacts' count."	Hidden Apex Test (Insert 50 contacts -> Assert Account field value).	Medium
Async Apex	"Create a Queueable class to make a callout to 'WarehouseDB'."	Hidden Apex Test using Test.setMock -> Assert Job Enqueued.[35]	High
LWC	"Build a component displaying a list of Contacts with a search bar."	DOM Testing (Jest) + Manual verification (LLM Judge on code structure).	High
SOQL	"Query all Accounts with at least one closed-won Opportunity."	Output Validation (Compare agent's query result with expected record set).	Low
API Integration	"Create a REST service to accept	Functional Test (Call endpoint via	High

	incoming Leads via JSON."	RestContext -> Assert Lead created).	

7. Infrastructure and DevOps Feasibility

Implementing this benchmark requires a robust DevOps pipeline. The core constraints are the Salesforce Org limits and the asynchronous nature of the platform.

7.1 Managing Limits and Pools

Dev Hubs have strict limits on Scratch Org creation. A standard Developer Edition Dev Hub allows only 6 scratch org creations per day, while Partner Business Orgs (PBOs) allow significantly more (up to 300/day).[30]

- **Strategy:** To scale the benchmark (e.g., running 100 evals), we must use **Org Snapshots**.[25] Snapshots allow creating a scratch org from a pre-configured template, skipping the lengthy metadata deployment phase.
- **Pooling:** Implement a service (like Hutte or a custom script) to maintain a pool of "warm" scratch orgs. The orchestrator checks out an org from the pool for a task and deletes it (or returns it to a dirty pile) after use.

7.2 Safety and Destructive Changes

Agents might attempt destructive operations (e.g., deleting a field or modifying the Admin profile).

- **Restriction:** The sf CLI supports destructiveChanges.xml. The benchmark environment must be isolated so that an agent cannot delete metadata in the Dev Hub itself.
- **Enforcement:** The containerization ensures the agent only has credentials for the Scratch Org. It never has access to the Dev Hub's auth token. The orchestrator handles the creation/deletion of orgs; the agent only "lives" within the org.[47]

8. Detailed Scenarios: Superbadge Case Studies

To prove the robustness of the framework, we detail how two specific Superbadges would be implemented as benchmark tasks.

8.1 Scenario A: Apex Specialist Superbadge (Developer)

This Superbadge tests heavy programmatic logic and integration.[33]

Benchmark Task 1: Warehouse Integration

- **Requirement:** Create a class WarehouseCalloutService that implements Queueable, makes a callout to https://th-superbadge-apex.herokuapp.com/equipment, and upserts

Product2 records.
- **Verification:**
  - **Static Analysis:** Check for implements Queueable, Http class usage, and upsert DML.
  - **Functional Test:** Inject a MockHttpResponseGenerator. Invoke the class. Query Product2 to verify records were created with correct fields (Cost, Lifespan, SKU).
  - **Governor Limit Check:** Ensure callout is not in a loop.

Benchmark Task 2: Maintenance Request Trigger

- **Requirement:** When a Case is closed (Type = Repair), create a new Case with Date_Due__c set to Today + min(Equipment_Cycle).
- **Verification:**
  - **Data Setup:** Create Vehicle, Equipment (Cycle = 15 days), and a Case.
  - **Action:** Update Case Status to 'Closed'.
  - **Assertion:** Query Case. Verify new Case exists. Verify Date_Due__c is exactly 15 days from today. Verify Vehicle__c matches parent.

8.2 Scenario B: Process Automation Specialist (Admin)

This Superbadge tests Flow and Declarative Logic.[12]

Benchmark Task 1: Automate Opportunities

- **Requirement:** Create a validation rule RB_High_Value_Opp preventing Closed Won if Amount > 100000 and Approved__c is false.
- **Verification:**
  - **Metadata Diff:** Check Opportunity.object for <validationRules> entry.
  - **Functional Test:**
    - Test A: Amount=150000, Approved=false, Stage=Closed Won. Assert: **Exception Thrown**.
    - Test B: Amount=150000, Approved=true, Stage=Closed Won. Assert: **Success**.

Benchmark Task 2: Flow for Robot Setup

- **Requirement:** Create a Process/Flow that creates a Robot_Setup__c record when an Opportunity is closed won. Calculate the Date__c as 180 days out, adjusting for weekends (Monday if Sat/Sun).
- **Verification:**
  - **Functional Test:** Close an Opportunity. Query Robot_Setup__c. Check if the Date__c formula logic correctly skipped the weekend. This tests the agent's ability to implement complex formula logic (CASE(MOD(...))) inside a Flow.

9. Conclusion

Building a Salesforce-oriented benchmark for AI requires shifting from simple code generation

metrics to a holistic "Platform Engineering" evaluation. The proposed **SF-AgentBench** integrates the rigidity of compiler checks (deployment validation), the functional assurance of unit testing (Apex tests), and the nuance of configuration analysis (metadata diffing).

By grounding the tasks in the official ADM-201 and PD1/PD2 curricula and utilizing the complexity of Superbadges, this benchmark provides a rigorous assessment of an agent's ability to act as a Salesforce professional. The reliance on sf CLI and Scratch Orgs ensures that the evaluation is realistic, reproducible, and safe. As agents evolve, this framework can expand to cover multi-cloud scenarios (Data Cloud, MuleSoft), positioning it as the definitive standard for Agentic Salesforce Development. This is not just a test of coding speed, but a test of architectural competence in a constraint-rich environment.

Citations

1

Appendix: Detailed Benchmark Task Specifications and Tooling Reference

A.1 Administrator Task Specifications (ADM-201 Deep Dive)

The Administrator role in Salesforce is pivotal. An AI agent must not only know "how" to configure a setting but "why" and "what impacts it has." The following sections detail specific benchmark tasks for the ADM-201 curriculum, focusing on areas that are notoriously difficult to automate or reason about.

A.1.1 Security and Sharing Architecture

This is the most frequent area of failure for both human and AI administrators. The benchmark must test the "Security Triangle": Profiles, Permission Sets, and Sharing Rules.

Task: "Private OWD and Sharing Rule Exception"

- **Prompt:** "Configure the Organization-Wide Defaults (OWD) for the 'Candidate__c' object to be Private. Create a Criteria-Based Sharing Rule that shares 'Candidate' records where 'Status__c' is 'Interviewing' with the 'Recruiters' Public Group with Read/Write access."
- **Complexity:** The agent must understand that OWD restricts access, while Sharing Rules open it up. It must create the Group, set the OWD (which involves a long-running background process in real life, but instant in Scratch Orgs for small data), and define the rule XML.

- **Verification:**
  - **Metadata Diff:** Verify Candidate__c.object-meta.xml contains <sharingModel>Private</sharingModel>.
  - **Functional Test:**
    1. Create User A (Recruiter) and User B (Standard).
    2. Insert Candidate (Status='New') as User B.
    3. Assert User A cannot see it.
    4. Update Status to 'Interviewing'.
    5. Assert User A *can* now see and edit it.
- **Insight:** This tests the agent's understanding of record-level security, a concept distinct from object-level permissions.[1]

A.1.2 Advanced Flow Orchestration

Flows are the visual coding language of Salesforce. The benchmark must verify the agent can construct valid Flow XML.

Task: "Screen Flow with Decision Logic and Fault Paths"

- **Prompt:** "Create a Screen Flow that allows a user to input a Case Subject and Priority. If Priority is 'High', create a Case immediately. If 'Low', create a Task for the owner. Include a Fault Path that sends an email to the Admin if the record creation fails."
- **Complexity:**
  - **Elements:** Screen, CreateRecords, Decision, Action (Email).
  - **Logic:** Branching based on input variables.
  - **Error Handling:** Connecting the "Fault" output of the Create element to the Email action. This is a best practice often ignored by simple code generators.
- **Verification:**
  - **Metadata Analysis:** Parse the flow-meta.xml. Check for <decisions> with <rules> corresponding to 'High' and 'Low'. Check for <faultConnector> on the RecordCreate element.
  - **Functional Test:**
    - Run Flow with Priority='High' -> Assert Case created.
    - Run Flow with Priority='Low' -> Assert Task created.
    - (Optional Advanced) Force an error (e.g., Validation Rule) and assert Email sent (mocked).

A.1.3 Data Management and Import

Agents are often tasked with migrating data. This requires handling CSV parsing, ID mapping, and order of operations.

Task: "Import Parent-Child Data (Accounts and Contacts)"

- **Prompt:** "Given two CSV files (accounts.csv, contacts.csv), import them into Salesforce.

The Contacts are related to Accounts via an 'External_ID__c' field."

- **Complexity:** The agent cannot simply insert Contacts; it must first insert Accounts, retrieve their Salesforce IDs (or use an External ID upsert), and then link the Contacts.
- **Tools Required:** sf data import bulk or sfdx-data-move-utility.
- **Verification:**
  - **Query:** SELECT Count() FROM Contact WHERE AccountId!= NULL.
  - **Validation:** Ensure the number of records matches the CSVs.
- **Insight:** This tests the agent's understanding of relational data integrity and the Bulk API.[28]

A.2 Developer Task Specifications (PD1/PD2 Deep Dive)

The Developer benchmark focuses on the programmatic layer. Here, efficiency and robustness are key.

A.2.1 Bulk Trigger Frameworks

Writing a trigger that works for one record is easy; writing one that works for 200 is hard.

Task: "Bulkified Opportunity Trigger"

- **Prompt:** "Write a trigger on Opportunity. When an Opportunity is closed won, create a 'Renewal' Opportunity. Ensure the code handles bulk updates of up to 200 records."
- **Failure Mode:** An agent might iterate Trigger.new and perform an insert inside the loop.
- **Verification:**
  - **Static Analysis (PMD):** Rule OperationWithLimitsInLoop will flag DML inside loops.
  - **Functional Test:**
    - Create 200 Opportunities.
    - Update all to 'Closed Won'.
    - Assert 200 Renewal Opportunities exist.
    - Check Limits.getDmlStatements() <= 1 (should be 1 bulk insert).
- **Rubric Score:**
  - Logic: 50%
  - Bulkification: 50% (Critical for passing).

A.2.2 Asynchronous Apex and Queueables

Handling long-running processes is a core PD2 skill.

Task: "Chained Queueable Callout"

- **Prompt:** "Create a Queueable class that performs a callout to an external API to fetch exchange rates. After updating currencies, chain another job to recalculate Opportunity amounts."

- **Complexity:**
  - **Interfaces:** Queueable, Database.AllowsCallouts.
  - **Chaining:** System.enqueueJob() inside the execute method.
  - **Testing:** Testing chained jobs requires Test.startTest() and Test.stopTest() but famously, you cannot chain jobs *inside* a test method unless you mock the behavior or use specific test patterns.
- **Verification:**
  - **Functional Test:** Mock the Callout. Run the test. Assert that the second job was enqueued (this is tricky to test directly, often requires checking a static flag or AsyncApexJob query in a separate context).
- **Insight:** This tests the agent's knowledge of Apex transaction boundaries and limits.[19]

A.2.3 Lightning Web Components (LWC)

The UI layer introduces JavaScript and DOM complexities.

Task: "Reactive Search Component"

- **Prompt:** "Build an LWC that displays a list of Contacts. Include a text input. When the user types, filter the list using the searchContacts Apex method. Use the @wire service."
- **Complexity:**
  - **Debouncing:** The agent should implement a delay (debounce) on the input to avoid spamming the server.
  - **Wire Service:** Correctly importing the Apex method and wiring it to the reactive property ($searchTerm).
  - **HTML:** Using template for:each to render the list.
- **Verification:**
  - **Jest Test:** The framework should run npm test on the component. The test should simulate input change and verify the wire adapter is updated.
  - **Static Analysis:** ESLint for LWC best practices (e.g., @api usage, no DOM manipulation outside lifecycle hooks).[21]

A.3 Benchmarking Environment Technical Specifications

To implement this benchmark, a sophisticated infrastructure is required. This section details the "Plumbing" of SF-AgentBench.

A.3.1 The Orchestrator

The Orchestrator is the control plane. It is a Python application responsible for scheduling and managing agent runs.

Responsibilities:

- **Queue Management:** Pulling tasks from the Task Registry.
- **Container Provisioning:** Spinning up Docker containers for the agent.
- **Org Allocation:** Interfacing with a "Pool Manager" to fetch a fresh Scratch Org credential.
- **Result Aggregation:** Collecting logs, test results, and metadata artifacts for scoring.

A.3.2 The Pool Manager

Creating a scratch org takes 2-10 minutes. This is too slow for an interactive benchmark. The Pool Manager maintains a buffer of "Ready" orgs.

Workflow:

1. **Cron Job:** Checks pool size (e.g., Target: 10 orgs).
2. **Creation:** If size < 10, executes sf org create scratch --definition-file config/superbadge-def.json.
3. **Pre-flight:** Installs common packages (e.g., Apex Specialist package 04t...).
4. **Tagging:** Marks the org as "Available".
5. **Checkout:** When the Orchestrator requests an org, the Pool Manager marks it "In Use" and returns the Auth URL.
6. **Cleanup:** Once the task is done, the org is deleted (sf org delete scratch). We do not recycle used orgs to ensure state purity.

A.3.3 The Agent-Computer Interface (ACI) Implementation

The ACI is the "API" the agent uses. It is implemented as a set of Python functions wrapped around the sf CLI.

Example Tool Implementation: deploy_metadata

Python

```
import subprocess
import json

def deploy_metadata(source_dir="force-app"):
    """
    Deploys metadata to the Salesforce Org.
    Returns structured success or error messages.
    """
    cmd = [
        "sf", "project", "deploy", "start",
```

```
        "--source-dir", source_dir,
        "--json"
    ]

    result = subprocess.run(cmd, capture_output=True, text=True)

    try:
        data = json.loads(result.stdout)
        if data['status'] == 0:
            return {"status": "success", "files_deployed": len(data['result']['files'])}
        else:
            # Structure the error for the Agent
            errors =
            if 'details' in data['result']:
                component_errors = data['result']['details'].get('componentFailures',)
                for err in component_errors:
                    errors.append(f"Error in {err['fileName']}: {err['problem']} (Line {err.get('lineNumber')})")
            return {"status": "failure", "errors": errors}
    except json.JSONDecodeError:
        return {"status": "error", "message": "Failed to parse CLI output"}
```

Insight: This wrapper converts the complex CLI JSON output into a simplified dictionary that the LLM can easily ingest, minimizing token usage and focusing the agent on the specific error logic (e.g., "Line 12: Variable undefined").

A.3.4 Security: The "Jail"

Running arbitrary code generated by an AI is dangerous. The Docker container must be locked down.

- **User:** Run as non-root user agent.
- **Filesystem:** Read-only root, writable /app/workspace.
- **Network:** Allow outbound only to *.salesforce.com and *.force.com. Block access to pypi.org or npm to prevent the agent from installing external libraries unless explicitly allowed by the task (e.g., LWC tasks might need npm install).

A.4 Scoring Rubrics and Metrics

The scoring system determines the benchmark's value. We employ a weighted scoring mechanism.

A.4.1 Metric Definitions

1. **Pass@1:** The percentage of tasks where the agent achieves a passing score (e.g.,

>80/100) on its first attempt.
2. **Resolution Rate:** The percentage of tasks solved after *n* turns of self-correction (e.g., Agent deploys -> Fails -> Reads Error -> Fixes Code -> Deploys -> Success). This measures the agent's debugging capability.
3. **Efficiency Score:**
    - Tokens used per task.
    - Number of CLI calls made. (An agent that blindly lists all metadata 10 times is inefficient).
    - Execution time.

A.4.2 Sample Rubric: "Process Automation Specialist - Challenge 1"

Task: Create a Validation Rule and Lead Assignment Rules.[12]

Criteria	Weight	Verification	Description
Validation Logic	30%	Functional Test	Insert Lead with invalid State. Assert error is thrown. Insert valid State. Assert success.
Formula Syntax	10%	Metadata Diff	Check Lead.object for correct use of CONTAINS and LEN in formula.
Assignment Rule	30%	Functional Test	Insert Lead with Source='Web'. Assert Owner is 'Rainbow Sales' Queue.
Queue Creation	20%	Metadata Diff	Verify Queues metadata exists for 'Rainbow Sales' and 'Assembly System'.
Safety	10%	Static Analysis	Ensure Validation Rule is Active. Ensure Assignment

			Rule does not have circular logic.

Total Score: 0-100.
Pass Threshold: 85.

A.4.3 Static Analysis Configuration (PMD)

The framework uses a custom ruleset.xml for PMD that targets Salesforce-specific anti-patterns.

Key Rules Enforced:

- ApexSOQLInjection: Detects unescaped user input in dynamic SOQL.
- OperationWithLimitsInLoop: The "Cardinal Sin" of Apex.
- AvoidDeeplyNestedIfStmts: Enforces code readability.
- AvoidGlobalModifier: Encourages public over global unless necessary (e.g., for Batch/RestResource).

A.5 Conclusion and Future Outlook

The **SF-AgentBench** framework provides the first comprehensive standard for evaluating Agentic AI in the Salesforce domain. By mirroring the complexity of the official Salesforce certification path and utilizing the robust tooling of the sf CLI and Scratch Orgs, it offers a realistic, safe, and rigorous testing ground.

As Salesforce evolves towards **Agentforce** and AI-driven development, this benchmark will serve as a critical yardstick. It ensures that the agents we build are not just capable of writing code, but capable of understanding the intricate, stateful, and governed environment of the Salesforce Platform. This moves the industry away from "Demo AI" that writes snippets, to "Enterprise AI" that delivers deployable, secure, and functional solutions.

Citations

[1]

Works cited

1. Salesforce Certified Platform Administrator Guide & Tips, accessed January 15, 2026, https://www.salesforceben.com/salesforce-administrator-certification/
2. Salesforce Platform Developer I (2025) – Comprehensive Study Guide - Nick Frates, accessed January 15, 2026, https://www.nickfrates.com/blog/salesforce-platform-developer-i-2025-comprehensive-study-guide

3. Top 10 Salesforce Trigger Interview Questions and Answers (2025) - Skillora AI, accessed January 15, 2026, https://skillora.ai/blog/top-salesforce-trigger-interview-questions-with-answers
4. Top 15+ Salesforce Interview Questions On Triggers - Hirist Blog, accessed January 15, 2026, https://www.hirist.tech/blog/top-15-salesforce-interview-questions-on-triggers/
5. Set Apex Class Access from Permission Sets - Salesforce Help, accessed January 15, 2026, https://help.salesforce.com/s/articleView?id=platform.perm_sets_setting_apex_access.htm&language=en_US&type=5
6. How to Use Custom Permissions in Apex - Salesforce Geek, accessed January 15, 2026, https://salesforcegeek.in/how-to-use-custom-permissions-in-apex/
7. Start Your Salesforce Admin Journey: ADM-201 Study Guide - Test-king.com, accessed January 15, 2026, https://www.test-king.com/blog/start-your-salesforce-admin-journey-adm-201-study-guide/
8. ValidationRule | Metadata API Developer Guide, accessed January 15, 2026, https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_validationformulas.htm
9. How to Use Validation Rules in Salesforce (+ Examples), accessed January 15, 2026, https://www.salesforceben.com/validation-rules-in-salesforce/
10. Flow | Metadata API Developer Guide, accessed January 15, 2026, https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_visual_workflow.htm
11. Testing Your Flow Before Activation - Salesforce Help, accessed January 15, 2026, https://help.salesforce.com/s/articleView?id=platform.flow_concepts_testing.htm&language=en_US&type=5
12. pkbparesh15/Process-Automation-Specialist-Solution - GitHub, accessed January 15, 2026, https://github.com/pkbparesh15/Process-Automation-Specialist-Solution
13. Profile | Metadata API Developer Guide, accessed January 15, 2026, https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_profile.htm
14. Salesforce Platform Developer 2 Study Guide | focusonforce.com, accessed January 15, 2026, https://focusonforce.com/courses/platform-developer-2-study-guide/
15. Salesforce Certified Platform Developer II - Trailhead Academy, accessed January 15, 2026, https://trailheadacademy.salesforce.com/certificate/exam-platform-dev2---Plat-Dev-301
16. Apex Trigger Scenario-Based Questions | by UATeam - Medium, accessed January 15, 2026, https://medium.com/@aleksej.gudkov/apex-trigger-scenario-based-questions-083bc437f5cd
17. Test Your Changes - Salesforce Help, accessed January 15, 2026,

https://help.salesforce.com/s/articleView?id=platform.testing_your_code.htm&language=en_US&type=5
18. Apex Testing: Tips for Writing Robust Salesforce Test Method - DZone, accessed January 15, 2026, https://dzone.com/articles/apex-testing-tips
19. 20+ Apex Interview Questions And Answers (2026) - S2 Labs, accessed January 15, 2026, https://s2-labs.com/blog/apex-interview-questions/
20. wallacelee/Data-Integration-Specialist-Superbadge - GitHub, accessed January 15, 2026, https://github.com/wallacelee/Data-Integration-Specialist-Superbadge
21. Unlock and Complete the Lightning Web Components Specialist Superbadge - Trailhead, accessed January 15, 2026, https://trailhead.salesforce.com/users/jimsharp/trailmixes/unlock-and-complete-the-lightning-web-components-specialist-supe
22. CristianoFIlho/SuperBadge-LWC - GitHub, accessed January 15, 2026, https://github.com/CristianoFIlho/SuperBadge-LWC
23. Architecture - SWE-agent documentation, accessed January 15, 2026, https://swe-agent.com/latest/background/architecture/
24. Scratch Orgs | Salesforce DX Developer Guide, accessed January 15, 2026, https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs.htm
25. Create a Scratch Org Based on a Snapshot | Salesforce DX Developer Guide, accessed January 15, 2026, https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_snapshots_create_scratch_org.htm
26. Build Your Own Scratch Org Definition File | Salesforce DX Developer Guide, accessed January 15, 2026, https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs_def_file.htm
27. Install Packages with the CLI | Salesforce DX Developer Guide, accessed January 15, 2026, https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_unlocked_pkg_install_pkg_cli.htm
28. sfdx force:data:tree:import | Salesforce Trailblazer Community - Trailhead, accessed January 15, 2026, https://trailhead.salesforce.com/trailblazer-community/feed/0D54V00007T40wOSAR
29. Salesforce Data Migration using SFDMU plugin | by Ayush chauhan | Medium, accessed January 15, 2026, https://medium.com/@ayushchauhan1999/salesforce-data-migration-using-sfdmu-plugin-85716b951ca6
30. Supported Scratch Org Editions and Allocations | Salesforce DX Developer Guide, accessed January 15, 2026, https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs_editions_and_allocations.htm
31. Scratch Org Allocations for Salesforce Partners | First-Generation Managed Packaging Developer Guide, accessed January 15, 2026,

https://developer.salesforce.com/docs/atlas.en-us.pkg1_dev.meta/pkg1_dev/isv_partner_scratch_org_allocations.htm
32. Improve handling of self-referencing lookups by force:data:tree commands #2364 - GitHub, accessed January 15, 2026, https://github.com/forcedotcom/cli/discussions/2364
33. Superbadge: Apex for Agentforce - Trailhead - Salesforce, accessed January 15, 2026, https://trailhead.salesforce.com/content/learn/superbadges/superbadge-apex-for-agentforce
34. Superbadge: Flow Fundamentals - Trailhead - Salesforce, accessed January 15, 2026, https://trailhead.salesforce.com/content/learn/superbadges/superbadge_flow_basics_sbu
35. pkbparesh15/Apex-Specialist-Solution: Solution of the ... - GitHub, accessed January 15, 2026, https://github.com/pkbparesh15/Apex-Specialist-Solution
36. giuseppetropea/salesforce-superbadge-apex-specialist - GitHub, accessed January 15, 2026, https://github.com/giuseppetropea/salesforce-superbadge-apex-specialist
37. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering - arXiv, accessed January 15, 2026, https://arxiv.org/pdf/2405.15793?
38. Take a Deep Dive into Metadata API Deployments | Salesforce Developers Blog, accessed January 15, 2026, https://developer.salesforce.com/blogs/2025/09/take-a-deep-dive-into-metadata-api-deployments
39. Master Metadata API Deployments with Best Practices | Salesforce Developers Blog, accessed January 15, 2026, https://developer.salesforce.com/blogs/2025/10/master-metadata-api-deployments-with-best-practices
40. Use Development and Application Lifecycle Workflow | Developer Guide | Salesforce Functions, accessed January 15, 2026, https://developer.salesforce.com/docs/platform/functions/guide/dev-alm-workflow.html
41. Salesforce Flow Tests: 5 Scenarios Where Apex Unit Tests Are the Better Choice, accessed January 15, 2026, https://lanefour.com/salesforce-admin/salesforce-flow-tests-5-scenarios-where-apex-unit-tests-are-the-better-choice/
42. Apex Tests for Flows | Salesforce Developer - YouTube, accessed January 15, 2026, https://www.youtube.com/watch?v=Nvs-Y2kKglw
43. PMD | Engines | Salesforce Code Analyzer, accessed January 15, 2026, https://developer.salesforce.com/docs/platform/salesforce-code-analyzer/guide/engine-pmd.html
44. scanner run (Retired) | Salesforce Code Analyzer v4 (Retired), accessed January 15, 2026, https://developer.salesforce.com/docs/platform/salesforce-code-analyzer/guide/run.html

45. The science of rubric design | Snorkel AI, accessed January 15, 2026, https://snorkel.ai/blog/the-science-of-rubric-design/
46. ResearchRubrics: A Benchmark of Prompts and Rubrics For Evaluating Deep Research Agents - arXiv, accessed January 15, 2026, https://arxiv.org/html/2511.07685v1
47. callawaycloud/sfdx-git-packager: Generates a "delta" metadata package based on the difference between two git refs (branches/commits) - GitHub, accessed January 15, 2026, https://github.com/callawaycloud/sfdx-git-packager
48. Apex Specialist Superbadge Trailmix - Trailhead, accessed January 15, 2026, https://trailhead.salesforce.com/users/lthomas/trailmixes/apex-specialist-superbadge
49. Easily load data into your Scratch Org (or Sandbox): Part 1 - Texei - Texei, accessed January 15, 2026, https://texei.com/en/advices/easily-load-data-into-your-scratch-org-or-sandbox-part-1/
50. Automating Salesforce Org Metadata Comparison With GitHub Actions, accessed January 15, 2026, https://www.salesforceben.com/automating-salesforce-org-metadata-comparison-with-github-actions/
51. scolladon/sfdx-git-delta: Generate the sfdx content in source format from two git commits, accessed January 15, 2026, https://github.com/scolladon/sfdx-git-delta
52. data Commands | Salesforce CLI Command Reference, accessed January 15, 2026, https://developer.salesforce.com/docs/atlas.en-us.sfdx_cli_reference.meta/sfdx_cli_reference/cli_reference_data_commands_unified.htm