

Terminal Bench: A Primer for Clinical Laboratory Professionals

What Problem Does Terminal Bench Solve?

Short answer: Testing AI agents the same way you test laboratory instruments.

Longer answer: When you validate a new analyzer or LIMS module, you run it against **known samples** with **expected results**. You document performance before deployment.

Terminal Bench does the same thing for AI agents.

The Lab Analogy

Traditional Instrument Validation:

1. Create test samples with known values
2. Run samples through instrument
3. Compare results to expected values
4. Calculate performance metrics (accuracy, precision, etc.)
5. Document for CAP inspection
6. Deploy if performance meets criteria

Terminal Bench for AI Agents:

1. Create test scenarios with known correct actions
2. Run agent through scenarios
3. Compare agent decisions to expected decisions
4. Calculate performance metrics (precision, recall, etc.)
5. Document for CAP inspection
6. Deploy if performance meets criteria

It's the same validation methodology, just for software agents instead of instruments.

Why Can't We Use Traditional Software Testing?

Traditional QC/QA testing assumes:

- Deterministic behavior (same input → same output)
- Known code paths
- Predictable edge cases

AI agents are different:

- Non-deterministic (same input → different outputs)
- Emergent behavior (not explicitly programmed)
- Unknown edge cases (can't predict all failures)

This is why we need a new testing framework.

Think of it like the difference between:

- **Chemistry analyzer:** Deterministic assay → traditional validation works
- **Pathologist:** Expert judgment → need peer review, proficiency testing, competency assessment

AI agents are more like pathologists than chemistry analyzers.

What Terminal Bench Provides

1. Standardized Task Format

Just like CAP proficiency testing has standardized sample formats, Terminal Bench has standardized task formats.

Example task structure:

- Scenario description (what the agent must do)
 - Input data (what the agent sees)
 - Success criteria (what counts as correct)
 - Evaluation method (how to score performance)

2. Realistic Workflows

Tasks model real-world scenarios, not toy problems.

Lab example:

- Not: "Classify this single result as normal/abnormal"
- Instead: "Triage a batch of 20 specimens with resource constraints"

3. Benchmark Library

A growing collection of validated tasks across domains.

Current Terminal Bench focus:

- Software engineering agents (coding tasks)
- Web navigation agents (browser automation)
- Tool-use agents (API interactions)

What's missing: Healthcare/clinical validation tasks

That's what you're building.

How Terminal Bench Works (Technical Overview)

Step 1: Define the Task

Create a scenario that tests agent capability.

Example: EDTA Contamination Detection

markdown

Scenario: Batch of 5 specimens may have EDTA contamination

Agent must:

- Analyze analyte patterns (K, Ca, Na, Cl, HCO₃, Glucose)
- Detect contamination signature (K↑ + Ca↓)
- Tune policy thresholds to minimize false positives/negatives
- Make HOLD/RELEASE decisions for each specimen

Success criteria:

- Precision > 80% (of HOLDS, how many are truly contaminated?)
- Recall > 50% (of actual contaminations, how many caught?)
- Respects batch constraint (max 2 HOLDS allowed)

Step 2: Prepare Test Data

Create specimens with known ground truth.

Example data:

json

```
{
  "specimen_id": "S101",
  "patient_id": "P002",
  "values": {
    "K": 6.5, // High (normal ~4.5)
    "Ca": 7.4, // Low (normal ~9.0)
    "Na": 137.4,
    "Cl": 101.3,
    "HCO3": 22.6,
    "Glucose": 127.5
  },
  "truth": {
    "label": "CONTAMINATION",
    "type": "EDTA_like"
  }
}
```

Step 3: Run the Agent

Give the agent access to:

- Scenario description
- Test data (without ground truth labels)
- Tools it needs (triage system, workflow policies)

Let it work.

Step 4: Evaluate Performance

Compare agent's decisions to ground truth.

Scoring:

```
python
```

True Positives: Agent said HOLD, specimen was contaminated

False Positives: Agent said HOLD, specimen was normal

False Negatives: Agent said RELEASE, specimen was contaminated

True Negatives: Agent said RELEASE, specimen was normal

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1 = 2 * (Precision * Recall) / (Precision + Recall)

Step 5: Document Results

Generate validation report for regulatory review.

Report includes:

- Task description
 - Agent configuration
 - Performance metrics
 - Pass/fail determination
 - Recommendation for deployment
-

Terminal Bench vs. Traditional Computer System Validation (CSV)

Aspect	Traditional CSV	Terminal Bench
What it tests	Deterministic code paths	Agent decision-making
Test cases	Functional requirements	Realistic scenarios
Pass criteria	100% specification compliance	Statistical performance thresholds
Documentation	Test plans, protocols, reports	Task definitions, metrics, results
Regulatory fit	Established (21 CFR Part 11)	Emerging (guidance pending)
Best for	Traditional LIMS/LIS modules	AI agents, agentic workflows

Key difference: CSV tests if software does what you told it to do. Terminal Bench tests if an agent makes good decisions.

Why This Matters for CLIA/CAP

Current State:

CAP GEN.43875: "Autoverification validation must include testing against known samples"

Problem: This was written for rule-based autoverification, not AI agents.

Questions CAP inspectors are starting to ask:

- "How do you validate an AI that makes non-deterministic decisions?"
- "What happens when the AI encounters a scenario you didn't test?"
- "How do you document AI decision-making for audit?"

Current lab approach: Apply CSV methodology (doesn't fit well)

With Terminal Bench:

You can answer:

- "We validated using Terminal Bench evaluation framework"
- "We tested against realistic scenarios with known ground truth"
- "Here's the task library we used: [GitHub link]"
- "Performance metrics: Precision 92%, Recall 87%, F1 89%"
- "Agent passed all contamination detection scenarios"

This maps to existing CAP language (known samples, performance metrics, documented validation) while adapting to AI characteristics.

The Two-Layer Model

Terminal Bench enables testing at two levels:

Layer 1: Baseline Correctness

Question: Can the agent execute deterministic rules correctly?

Example task: EDTA contamination detection

- Clear signature ($K \uparrow + Ca \downarrow$)
- Known thresholds
- Binary decision (HOLD/RELEASE)

Finding: Strong agents (GPT-4o, Sonnet-4.5) solve this 100%

Validation purpose: Proves agent understands domain

Layer 2: Workflow Judgment

Question: Can the agent make good decisions under uncertainty?

Example task: Sequential triage with resource constraints

- Incomplete information (results arriving over time)
- Resource limits (review capacity, turnaround time)
- Competing objectives (safety vs. cost vs. speed)
- Irreversible decisions

Finding: TBD (this is the validation frontier)

Validation purpose: Proves agent can operate safely in production

For CAP compliance, you need BOTH layers validated.

Harbor: The Evaluation Runtime

Terminal Bench = Task format/standard

Harbor = Software that runs Terminal Bench tasks

Think of it like:

- **HL7** = Message standard
- **Interface engine** = Software that processes HL7 messages

Harbor provides:

- Task execution environment (containerized, reproducible)
- Agent interaction layer (gives agent access to tools)
- Scoring/evaluation engine (calculates metrics)
- Results reporting (generates documentation)

You don't need to build your own evaluation infrastructure - Harbor handles it.

How This Integrates with Your Current Validation Process

Traditional LIS Module Validation:

- Step 1: Requirements specification
- Step 2: Test protocol development
- Step 3: Installation qualification (IQ)
- Step 4: Operational qualification (OQ)

Step 5: Performance qualification (PQ)

Step 6: Ongoing monitoring

Adding AI Agent Validation:

Step 1: Requirements specification (same)

Step 2: Terminal Bench task selection/creation ← NEW

Step 3: Installation qualification (same)

Step 4: Operational qualification (same)

Step 5: Agent performance evaluation (Terminal Bench) ← NEW

Step 6: Ongoing monitoring + periodic re-evaluation ← ENHANCED

Terminal Bench fits BETWEEN traditional OQ and PQ.

It doesn't replace existing validation - it augments it for AI-specific characteristics.

Real-World Example: LIS Autoverification Agent

Traditional Validation (Rule-Based System):

Test Case 1: Normal K result (4.2 mmol/L)

Expected: Auto-release

Actual: Auto-release

Result: PASS

Test Case 2: Critical K result (6.8 mmol/L)

Expected: Hold for review

Actual: Hold for review

Result: PASS

[... 50 more test cases ...]

Conclusion: System performs per specification

Problem: These test cases don't tell you how the AI will behave on NEW scenarios.

Terminal Bench Validation (AI Agent):

Task: EDTA Contamination Detection

- 100 specimens with varying contamination patterns
- Agent must detect signature and tune thresholds

- Performance measured across entire task set

Results:

- Precision: 94% (6% false positive rate)
- Recall: 89% (11% missed contaminations)
- F1: 91.4%
- Resource constraint compliance: 100%

Conclusion: Agent demonstrates competence on realistic scenarios

This tells you how the AI performs across a DISTRIBUTION of cases, not just individual test points.

Getting Started with Terminal Bench

For Lab Directors:

Before deploying AI autoverification, ask your vendor:

1. "What Terminal Bench tasks have you validated against?"
2. "Can I see the task definitions and performance metrics?"
3. "How often do you re-evaluate as the AI is updated?"
4. "Can we run your agent against our own custom scenarios?"

If they can't answer these, their validation is incomplete.

For LIS Vendors:

To validate your AI agent:

1. Select relevant Terminal Bench tasks (or create custom ones)
2. Run your agent through the task library
3. Document performance metrics
4. Include in your validation package
5. Re-run when AI is updated

This creates audit-ready documentation that CAP inspectors understand.

For QA Managers:

To document AI validation for CAP:

1. Maintain task library (Terminal Bench scenarios)

2. Document agent performance on each task
3. Track performance over time (ongoing monitoring)
4. Update validation when:
 - AI model changes
 - New failure modes discovered
 - Workflow changes
 - Regulatory guidance updates

This maps to CAP's lifecycle validation requirements.

Common Questions

Q: Is Terminal Bench FDA/CAP approved?

A: Terminal Bench is an evaluation methodology, not a product requiring approval.

It's like asking "Is proficiency testing FDA approved?" - it's a validation approach.

CAP doesn't prescribe specific validation methods, just requirements (accuracy, precision, documented performance). Terminal Bench is one way to meet those requirements.

Q: Can I use Terminal Bench for FDA submissions?

A: Yes, with appropriate documentation.

FDA guidance on "Clinical Decision Support Software" requires validation that the AI performs as intended. Terminal Bench provides that evidence.

You'd include:

- Task definitions
- Performance metrics
- Pass/fail criteria
- Results documentation

Q: Do I need to hire AI experts to use this?

A: No more than you need PhD chemists to validate a chemistry analyzer.

You need:

- Domain expertise (lab workflows, CLIA/CAP requirements) ✓ You have this
- Task library (pre-built Terminal Bench scenarios) ← Being developed

- Evaluation runtime (Harbor) ← Open source software

The methodology is accessible to existing lab QA teams.

Q: How is this different from "AI validation" services from big consulting firms?

A: Most AI validation services focus on:

- Model accuracy metrics (sensitivity, specificity)
- Bias detection
- Data quality assessment

Terminal Bench focuses on:

- Workflow-level behavior
- Realistic scenario performance
- Decision-making under constraints

Both are needed. Terminal Bench addresses the gap that traditional AI validation misses: **how does the agent actually perform in your workflow?**

Q: What if Terminal Bench tasks don't exist for my use case?

A: You can create custom tasks.

This is actually an advantage - you validate against YOUR specific workflows, not generic scenarios.

Task creation requires:

1. Scenario definition (what should the agent do?)
2. Test data (specimens with known ground truth)
3. Success criteria (what counts as correct?)
4. Evaluation logic (how to score performance)

For regulated environments, custom task creation is often necessary to match your specific risk profile.

Where to Learn More

Terminal Bench Project:

- Website: <https://www.tbench.ai>
- Paper: <https://arxiv.org/abs/2501.11868>

- GitHub: <https://github.com/laude-institute/terminal-bench>

Harbor Evaluation Runtime:

- GitHub: <https://github.com/laude-institute/harbor>
- Documentation: [Coming soon]

LIS AI Validation Hub (This Project):

- Task library for clinical labs
 - Validation methodologies for CLIA/CAP compliance
 - Integration with MCP architecture
 - GitHub: [Your repo link]
-

Next Steps

If you're responsible for LIS AI validation:

1. **Read the Terminal Bench paper** (30 min)
 - Understand the methodology
 - See example tasks from software domain
2. **Review the EDTA contamination task** (15 min)
 - First Terminal Bench task for clinical labs
 - Shows how methodology applies to LIS
3. **Assess your current validation approach** (1 hour)
 - Are you testing individual cases or realistic workflows?
 - Do you measure performance across scenario distributions?
 - Can you explain agent behavior under uncertainty?
4. **Plan Terminal Bench integration** (ongoing)
 - Identify high-risk workflows requiring validation
 - Create or select relevant tasks
 - Establish performance thresholds
 - Document for regulatory review

The sooner you start thinking in Terminal Bench terms, the better prepared you'll be when CAP inspectors start asking AI-specific validation questions.

This primer is part of the LIS AI Validation Hub, a project to adapt software agent evaluation methodologies for clinical laboratory compliance requirements.

For questions, clarifications, or to contribute tasks: [Your contact/GitHub]