Technical Architecture Assessment: DBT vs PySpark for Regulatory Calculation Platform

Executive Summary

This assessment evaluates whether to maintain our current PySpark-based regulatory calculation platform or introduce DBT for certain data processing tasks. While DBT is a capable SQL transformation tool that can meet basic regulatory requirements like attribute-level lineage, introducing it would fragment our architecture and sacrifice the significant investments we've made in our current platform.

Our PySpark implementation with wrapped operators provides row-level lineage—tracking exactly which rows contributed to each result. This capability, impossible to achieve with SQL-based tools, provides competitive advantage in regulatory compliance and positions us for AI/LLM integration.

After comprehensive analysis across seven critical dimensions, the results are decisive:

- **Current PySpark Solution**: 85.5% weighted score
- **DBT Alternative**: 43.0% weighted score

Recommendation: Continue enhancing the existing PySpark platform. Maintain architectural consistency to leverage existing investments in technology, expertise, and operational procedures while preserving our competitive advantages.

Key Differentiators: - Both meet attribute-level lineage (CCAR requirement) - Row-level tracing provides competitive advantage beyond requirements - Direct SME control over calculations through unified platform - Architectural consistency leveraging existing investments - Mixed-squad operational model drives organizational innovation

Assessment Methodology

We evaluated both solutions across seven weighted criteria, with weights determined by business criticality and regulatory requirements. Scoring is based on measurable capabilities and documented limitations.

Evaluation Results

Primary Criteria (90% of total weight)

Criteria	Weight	PySpark	DBT	Key Differentiator
Regulatory Compliance	25%	9/10	4/10	Both provide attribute-level lineage for CCAR. PySpark adds row-level tracing showing exact source contributions—a significant competitive advantage.
Business User Empowerment	25%	9/10	2/10	Current platform enables mixed squads with row-level assurance for SME confidence. DBT enforces traditional separation without granular validation capabilities.
Future-Ready Architecture	20%	9/10	3/10	Python's complete ecosystem enables LLM integration. DBT's SQL/Jinja templates complicate AI code generation. Row-level tracing essential for validating AI outputs.
Scalability	15%	9/10	6/10	Spark designed for distributed computing of complex calculations. DBT relies on warehouse compute, suitable for SQL but not algorithmic workloads.

Secondary Criteria (10% of total weight)

Criteria	Weight	PySpark	DBT	Key Differentiator
Development Velocity	5%	8/10	6/10	Complex regulatory logic requires full programming capabilities. DBT's SQL/Jinja approach cannot handle sophisticated mathematical models.
Total Cost of Ownership	3%	7/10	6/10	Maintaining one technology stack vs managing two. DBT would require additional

Criteria	Weight	PySpark	DBT	Key Differentiator
				training, tooling, and operational overhead.
Team Expertise	2%	8/10	5/10	Python dominance in regulatory calculations vs DBT's SQL focus

Weighted Results

- **PySpark**: 8.55/10 (85.5%)
- **DBT**: 4.30/10 (43.0%)

Risk Analysis

Current Solution (PySpark)

Strengths: - Unified technology stack with consistent practices across all data processing - Industry-standard technology for regulatory calculations (used by major banks) - Proven platform with established team expertise and operational procedures - Comprehensive audit trail through row-level lineage exceeding regulatory requirements - Enables rapid regulatory response (hours vs weeks) - Supports mixed-squad organizational model - Leverages Python's vast ecosystem of financial and mathematical libraries - Significant existing investment in platform, training, and processes

Risks & Mitigations: - **Custom solution maintenance** → Mitigated by established team expertise and extensive Python ecosystem - **Knowledge concentration** → Addressed through documentation and Python's widespread use in finance - **Enterprise standards** → Resolved via API integration layer while maintaining consistency

DBT Alternative

Critical Risks: - **Row-level tracing gap**: Cannot provide row-level lineage showing exact source row contributions - **Future regulatory exposure**: No granular tracing to anticipate evolving requirements - **Business agility loss**: Returns to multi-week development cycles - **Organizational regression**: Eliminates mixed-squad benefits - **Trust deficit**: No row-level assurance for SME-built or LLM-generated calculations - **Architectural fragmentation**: Introduces inconsistency after significant platform investment

DBT's Strengths in Isolation: - Strong SQL-based transformation patterns for data warehousing - Built-in testing and documentation for data pipelines - Active community and vendor support - Good for ETL/ELT workflows where SQL is sufficient - Can provide attribute-level lineage through various tools - Well-established in data engineering community

However: Introducing it alongside our existing PySpark platform would: - Fragment our unified architecture - Create operational complexity - Sacrifice the benefits of consistency - Provide no additional capabilities we currently lack

Strategic Analysis

Technology Suitability and Architectural Consistency

The comparison between Python/PySpark and DBT reveals tools designed for different purposes, each with their strengths:

Python/PySpark: Complete Programming Ecosystem - Established as the dominant language for financial services (JPMorgan's Athena, Goldman Sachs' GS Quant) - Provides full programming capabilities essential for complex mathematical models - Extensive ecosystem including NumPy, Pandas, and specialized quantitative finance packages - Enables implementation of sophisticated algorithms required by Basel III and CCAR - Our wrapped operators add row-level lineage capabilities beyond standard PySpark

DBT: SQL Transformation Framework - Excellent for data warehouse transformations and ETL/ELT workflows - Provides attribute-level lineage capabilities through various tools - Strong community and vendor support - Template-based approach using SQL and Jinja - Well-suited for standard reporting and data transformations

However, introducing DBT alongside our existing PySpark platform would: - Fragment the architecture into two separate technology stacks - Require maintaining different deployment, monitoring, and debugging approaches - Split team expertise and knowledge - Create gaps in our row-level lineage coverage - Add operational complexity without providing capabilities we lack

Given our significant investment in the PySpark platform with wrapped operators, maintaining architectural consistency provides greater value than introducing technology diversity.

Row-Level Tracing: Beyond Compliance to Competitive Advantage

While both platforms can meet CCAR's attribute-level lineage requirements, our platform's row-level tracing capability represents a strategic differentiator that:

1. **Exceeds Regulatory Requirements**: While attribute-level lineage satisfies CCAR, row-level tracing provides granular visibility that:
    - Shows exactly which rows contributed to each calculated value
    - Enables precise debugging of calculation issues
    - Provides complete audit trails for regulatory inquiries
    - Positions us ahead of evolving regulatory requirements
2. **Enables SME Autonomy**: When business users directly build calculations, they need granular visibility to:
    - Validate their logic works correctly on specific data points
    - Understand calculation behavior on edge cases
    - Make targeted adjustments with confidence
    - Debug issues independently without technical support
3. **Provides LLM Assurance**: As AI generates increasingly complex calculations:
    - Row-level tracing becomes **mandatory** for trust and validation
    - SMEs can verify AI-generated logic on actual data points
    - Immediate feedback loops improve LLM outputs
    - Granular audit trails satisfy both regulatory and risk management needs
4. **Supports Real-Time Analysis**: Organizations can:

- Trace the impact of changes at individual record level
- Quickly identify and correct calculation errors
- Provide detailed explanations to regulators
- Maintain confidence in rapidly evolving calculations

This capability, built into our wrapped operators, cannot be replicated in any SQL-based tool and represents a significant competitive advantage.

Operating Model Impact

Current: Mixed-Squad Innovation

The current platform enables a revolutionary operating model:

```
SME identifies need → Direct platform interaction → Immediate
validation → Production
(Hours)
```

Organizational Benefits: - Cross-functional teams with shared ownership - Real-time collaboration between domains - Elimination of handoff bottlenecks - Continuous learning and adaptation

DBT: Traditional Separation

DBT enforces conventional organizational boundaries:

```
SME documents → Technical interpretation → Development → Testing →
Review → Production
(Weeks)
```

Organizational Limitations: - Siloed teams with handoff dependencies - Documentation-based communication - Technical team becomes bottleneck - Limited domain expert involvement

Future Technology Integration

Original Vision Realized

The platform's founding vision—empowering SMEs to directly build regulatory calculations—required a fundamental rethinking of traditional IT/business boundaries. Row-level tracing emerged as the critical enabler of this vision, providing the guaranteed assurance method necessary for non-technical users to confidently create and modify complex calculations.

This architectural decision now proves prescient as Large Language Models transform software development. What was originally a trust mechanism for human SMEs becomes **mandatory infrastructure** for AI-generated calculations.

LLM-Enhanced Capabilities

The emergence of AI-assisted development fundamentally changes the evaluation:

PySpark + LLM Integration: - SMEs interact directly: “Create Basel III calculation with SME adjustment” - AI generates code within existing wrapper framework - **Row-level tracing provides mandatory validation** of AI-generated logic - SMEs verify calculations work correctly on specific data points - Immediate feedback loops improve LLM accuracy for regulatory domain - Technical team focuses on platform capabilities, not calculation debugging - Maintains consistency with existing architecture

DBT + LLM Challenges: - Jinja templating complicates AI code generation - Technical review still required for all changes - Limited SME interaction with AI tools - Continued dependency on technical intermediation

Implementation Paths

Option 1: Platform Enhancement (Recommended)

Scope: Extend current capabilities while preserving advantages

Key Initiatives: 1. **AI Integration** (Q1-Q2) - Implement Copilot with wrapper libraries - Natural language calculation interface - Leverage Python’s dominance in quantitative finance

2. **Developer Experience** (Q2-Q3)
    - Enhanced VS Code integration
    - Automated documentation generation
3. **Operational Excellence** (Q3-Q4)
    - Expanded monitoring and observability
    - Performance optimization framework

Investment: $2-3M over 12 months **Risk**: Low (incremental improvements) **Return**: Maintains competitive advantage while modernizing

Option 2: DBT Migration (Not Recommended)

Scope: Complete platform replacement

Requirements: - Rebuild UI calculation builder - Develop custom row-level tracing - Migrate calculation library - Retrain all users - Reorganize teams

Investment: $8-12M over 24 months **Risk**: High (technical and organizational) **Return**: Negative (loses competitive advantages)

Option 3: Hybrid Approach (Alternative)

Scope: Selective DBT adoption for non-regulatory workloads

Implementation: - Maintain PySpark for regulatory calculations - Evaluate DBT for standard ETL processes - Develop integration layer - Gradual workload migration

Investment: $4-6M over 18 months **Risk**: Medium (integration complexity) **Return**: Limited (architectural diversity without core benefits)

Recommendations

Primary Recommendation

Continue with PySpark platform enhancement focusing on:

1. **Immediate Actions** (0-3 months)
   - Form LLM integration task force
   - Document current platform advantages
   - Establish enhancement roadmap
2. **Short-term Goals** (3-9 months)
   - Deploy AI-assisted calculation tools
   - Expand SME self-service capabilities
   - Implement automated compliance checks
3. **Long-term Vision** (9-18 months)
   - Full natural language calculation platform
   - Predictive regulatory change management
   - Industry-leading compliance automation

Success Metrics

- **Regulatory Response Time**: Maintain <24 hour implementation
- **SME Autonomy**: >80% calculations managed directly
- **Compliance Coverage**: 100% row-level traceability
- **Platform Adoption**: >95% user satisfaction

Conclusion

The assessment clearly demonstrates that the current PySpark platform provides superior value across all critical dimensions. While DBT can meet basic regulatory requirements like attribute-level lineage, it cannot provide the row-level tracing that represents our competitive advantage.

The fundamental mismatch between DBT's SQL transformation capabilities and the mathematical requirements of regulatory calculations makes it unsuitable for our core use cases. More importantly, having invested significantly in:

- **Wrapped operators with row-level lineage**: A sophisticated architecture that tracks exact row contributions
- **Team expertise**: Specialized knowledge in Python/PySpark and our custom framework
- **Operational procedures**: Established monitoring, deployment, and debugging practices
- **Mixed-squad model**: Organizational innovation that empowers SMEs

Introducing DBT would fragment this investment and create architectural inconsistency without providing compensating benefits.

Beyond meeting CCAR's attribute-level lineage requirements, the platform's row-level tracing capability represents a strategic investment in:

- **Regulatory Future-Proofing**: Anticipating the inevitable evolution toward more granular requirements
- **SME Empowerment**: Providing the assurance needed for direct business user control
- **AI Readiness**: Enabling mandatory validation for LLM-generated calculations
- **Competitive Differentiation**: Offering capabilities that exceed industry standards

The recommended enhancement path preserves these strategic advantages while addressing modernization needs through targeted improvements. By maintaining architectural consistency with our existing PySpark platform, we: - Leverage existing investments - Avoid operational complexity - Maintain competitive advantages - Position for future innovation

This approach ensures continued leadership in regulatory technology innovation while building on proven capabilities rather than fragmenting the architecture.

Appendix: Criteria Definitions

Regulatory Compliance: Ability to provide complete audit trails, data lineage, and meet all regulatory reporting requirements.

Business User Empowerment: Capability for domain experts to directly create and modify calculations without technical intermediation. Includes providing abstraction layers and UI tools that allow SMEs to work with Python-based calculations through business-friendly interfaces, with full row-level traceability for confidence.

Future-Ready Architecture: Platform compatibility with emerging technologies, particularly AI/ML and natural language interfaces. Ability to leverage the complete ecosystem of a programming language versus template-based approaches. Includes maintaining architectural consistency to build on existing capabilities.

Scalability: Ability to handle exponential growth in calculations, data volumes, and computational requirements. Includes distributed processing of complex mathematical models and maintaining consistent scaling patterns across all data processing needs.

Development Velocity: Speed of implementing new requirements from identification to production deployment. Includes ability to implement complex mathematical models and algorithms required for regulatory calculations, as well as maintaining consistency with existing codebase and practices.

Total Cost of Ownership: Comprehensive costs including development, operations, training, maintenance, and the cost of architectural complexity. Reflects the efficiency of maintaining one technology stack versus the overhead of managing multiple systems.

Team Expertise: Alignment with current team skills, leveraging existing platform knowledge, and maintaining consistency with established practices. Reflects the value of existing investments in training, documentation, and operational procedures.