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 migrate to a DBT-based architecture. The comparison reveals fundamental differences in technological capabilities: Python/PySpark provides a complete programming ecosystem essential for complex regulatory calculations, while DBT offers SQL-based transformations suited for ETL workflows but lacking the mathematical capabilities regulatory calculations require.

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: Enhance the existing PySpark platform rather than migrate. The current solution provides superior regulatory compliance, business agility, and future readiness. Migration would introduce significant risks while eliminating competitive advantages.

Key Differentiators: - Attribute-level lineage meets CCAR requirements - Row-level tracing provides competitive advantage and LLM assurance - Direct SME control over calculations - 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	Attribute-level lineage (CCAR requirement) + row-level tracing (competitive advantage). DBT provides model-level lineage only; row-level tracing would require extensive custom development.
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.

Criteria	Weight	PySpark	DBT	Key Differentiator
Total Cost of Ownership	3%	7/10	6/10	Python developers are readily available in financial services. DBT would require retraining teams and hiring new skills for a tool unsuited to the core use case.
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: - Industry-standard technology for regulatory calculations (used by JPMorgan, Goldman Sachs, etc.) - Proven platform with established team expertise - Comprehensive audit trail for regulatory compliance - Enables rapid regulatory response (hours vs weeks) - Supports mixed-squad organizational model - Leverages Python's vast ecosystem of financial and mathematical libraries

Risks & Mitigations: - **Custom maintenance burden** → Mitigated by Python's vast ecosystem and widespread expertise in financial services - **Specialized knowledge** → Python skills are common in regulatory calculation teams; extensive documentation exists - **Enterprise standards** → Resolved via API integration layer

DBT Alternative

Critical Risks: - **Regulatory compliance gap**: Cannot provide attribute-level lineage required by CCAR - **Future regulatory exposure**: No row-level 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 - **Migration complexity**: 18-24 month timeline with uncertain outcomes

Limited Benefits: - Provides SQL-based transformation patterns familiar to data engineers - Offers built-in testing and documentation for data pipelines - Vendor-supported solution with active community - Well-suited for standardized ETL/ELT workflows - However, lacks capabilities for complex regulatory calculations

Strategic Analysis

Technology Suitability: Purpose-Built vs Template-Based

The comparison between Python/PySpark and DBT is not about technological advancement—both are modern tools serving different purposes. The critical distinction lies in their fundamental design philosophies and resulting capabilities for regulatory calculations.

Python/PySpark: Complete Programming Ecosystem - Python has established itself as the dominant language for financial services, with JPMorgan's Athena platform and Goldman Sachs' GS Quant built on Python - Major financial institutions use Python for quantitative research, risk management, and regulatory compliance - Provides full programming language capabilities essential for complex mathematical models - Extensive ecosystem including NumPy, Pandas, and specialized packages for quantitative finance - Enables implementation of sophisticated algorithms required by Basel III and CCAR - Natural fit for teams already using Python for risk modeling and analytics

DBT: SQL Transformation Framework - Designed for ETL/ELT workflows, not computational workloads - Primarily used in financial services for basic reporting and data warehouse management - Template-based approach using SQL and Jinja - Excellent for standardized transformations but fundamentally unable to implement Basel III mathematical models - Cannot perform the complex calculations required for risk-weighted assets, CVA, or other regulatory metrics - No evidence of adoption for regulatory calculations in any major financial institution

The talent pool reality strongly favors Python: financial institutions employ thousands of Python developers for quantitative work, while DBT expertise centers around data engineering and analytics engineering roles. Python is widely used in quantitative finance for complex calculations and risk management. For regulatory calculation teams specifically, Python skills are the industry standard—every major bank's quantitative teams use Python as their primary language.

Row-Level Tracing: Beyond Compliance to Competitive Advantage

While CCAR mandates attribute-level lineage, our platform's row-level tracing capability represents a strategic differentiator that:

1. **Anticipates Future Regulations**: Regulatory requirements consistently evolve toward greater granularity and transparency. Row-level tracing positions us ahead of this curve.

2. **Enables SME Autonomy**: When business users directly build calculations, they need granular visibility to:

    - Validate their logic works correctly
    - Understand calculation behavior on specific cases
    - Make targeted adjustments with confidence
    - Debug edge cases independently

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 Adjustments**: 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

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

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. The fundamental mismatch between DBT's SQL transformation capabilities and the mathematical requirements of regulatory calculations makes migration technically inappropriate, not merely inadvisable.

Python has established itself as the industry standard for financial calculations, with every major financial institution using it for risk management and regulatory compliance. Our platform leverages this ecosystem while adding competitive advantages through row-level tracing and mixed-squad enablement.

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 mixed-squad operational model this platform enables represents genuine organizational innovation. In an era where AI will increasingly generate complex calculations, row-level tracing transforms from a “nice-to-have” to an essential trust mechanism.

While DBT excels at SQL-based data transformations, adopting it for regulatory calculations would: - Fail to support the complex mathematical models these calculations require - Meet only minimum regulatory requirements without future-proofing - Eliminate the row-level assurance critical for LLM adoption - Reduce organizational agility and innovation - Remove established competitive advantages - Require significant investment to achieve inferior capabilities

The recommended enhancement path preserves these strategic advantages while addressing modernization needs through targeted improvements. This approach positions the organization for continued leadership in regulatory technology innovation, particularly as AI transforms how calculations are created and validated.

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.

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.

Scalability: Ability to handle exponential growth in calculations, data volumes, and computational requirements. Includes distributed processing of complex mathematical models, not just data transformation.

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.

Total Cost of Ownership: Comprehensive costs including development, operations, training, and opportunity costs.

Team Expertise: Alignment with current team skills, availability of talent in the market, and prevalence of the technology in regulatory calculation contexts. Python dominates financial services and regulatory calculations, while DBT is primarily used for data transformation workflows.