GSoC 2025 Project Proposal: Enhancing Dubbo Python Serialization

Personal Information

Name: Aditya Yadav
Email: adiworkprofile@gmail.com
Phone: +91-8920735656
Location: New Delhi, India
GitHub: github.com/aditya0yadav
LinkedIn: linkedin.com/in/2580aditya

Academic Background

University: Indian Institute of Technology Madras
Degree: Bachelor's in Data Science and Mathematics
Expected Graduation: June 2026

Project Overview

Project Title: Enhancing Dubbo Python Serialization
Organization: Apache Software Foundation (Dubbo)
Mentor: Albumen Kevin (albumenj@apache.org), Apache Dubbo PMC

Executive Summary

I propose to develop a robust, built-in serialization layer for Dubbo-Python using Pydantic, addressing a critical usability gap between Dubbo's Java and Python implementations. This enhancement will significantly simplify development workflows, reduce error-prone manual serialization, and provide automatic data validation—ultimately bringing the Python implementation closer to feature parity with Dubbo-Java.

Problem Statement

Unlike Dubbo-Java, which provides built-in serialization options, Dubbo-Python currently requires developers to manually implement serialization and deserialization for each service. This creates several pain points:

1. Increased development overhead with repetitive boilerplate code
2. Higher risk of serialization errors and inconsistencies
3. Lack of automatic validation, leading to potential runtime issues

4. Fragmented approaches across projects using Dubbo-Python

Proposed Solution

I will implement a comprehensive serialization layer that:

1. Leverages Pydantic for robust data modeling and validation
2. Supports multiple formats (JSON and Protobuf) with a unified API
3. Integrates seamlessly with existing Dubbo-Python architecture
4. Provides an intuitive developer experience with minimal configuration

Technical Architecture

The solution will introduce a `PydanticSerialization` class that will:

1. Handle the conversion between Pydantic models and serialized formats
2. Integrate directly into `RpcMethodHandler` and `Client`/`Server` classes
3. Provide automatic validation during serialization and deserialization
4. Support both synchronous and asynchronous patterns

Qualifications & Experience

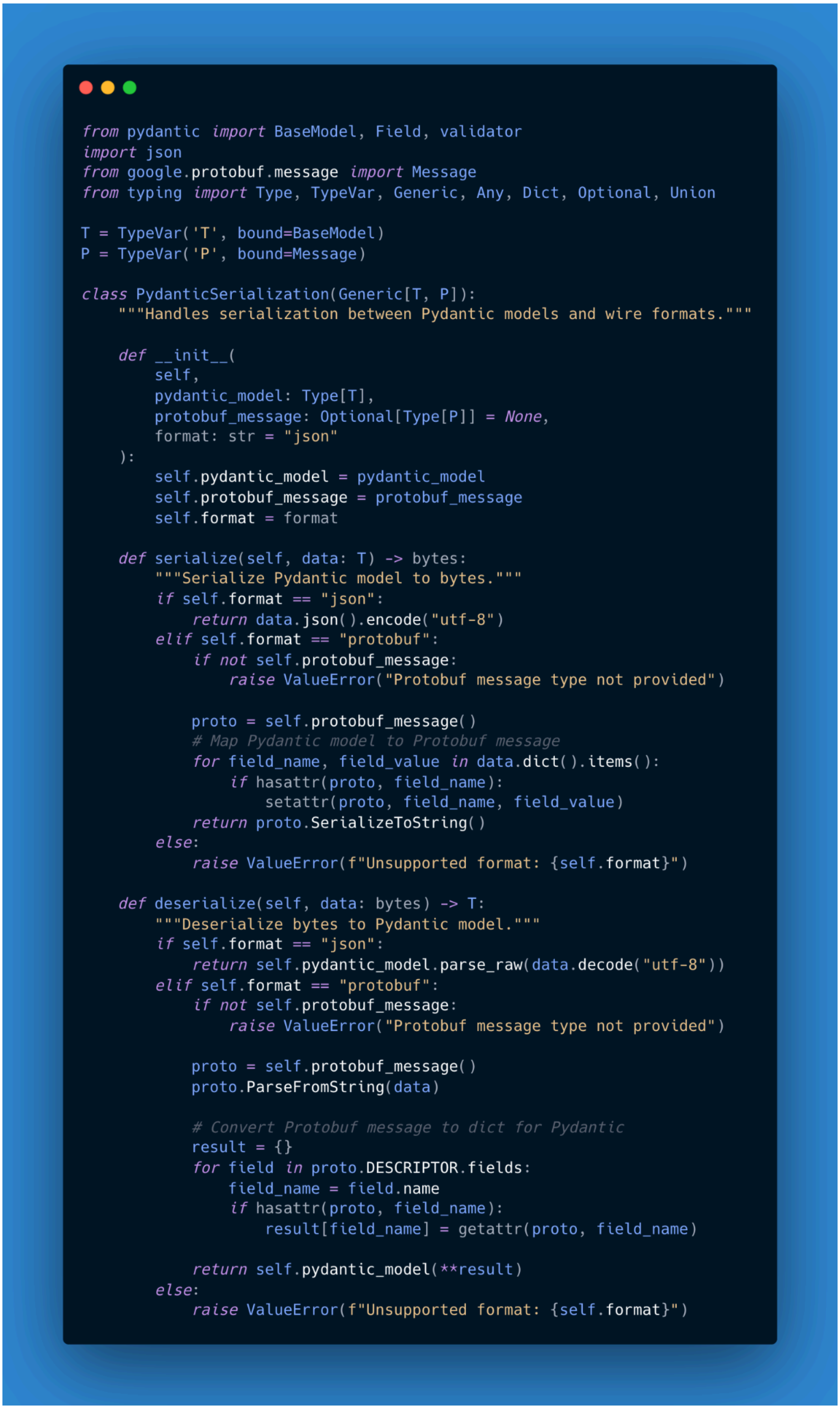
My background uniquely positions me to deliver this project successfully:

Relevant Technical Experience

- **Software Engineer Intern at AcutusAI** (8 months): Built production-grade synthetic data generation systems in Python
- **Apache Beam Contributor**: Implementing OpenAI API vector embeddings with comprehensive testing (branch link)
- **Apache Airflow Contributor**: Submitted PRs enhancing Python-based workflow orchestration
- **Freelance Developer**: Delivered 5+ MERN and Python-based web applications for clients
- **Research Publication**: "Rust vs. C++ Performance" (Feb 2025) – Top-ranked paper at IIT Madras benchmarking system programming efficiency

Skills Relevant to This Project

- Advanced Python programming with deep knowledge of data modeling
- Experience with RPC frameworks and distributed systems architecture
- Strong understanding of serialization formats (JSON, Protobuf)
- Prior contributions to Apache projects and open-source ecosystems
- Background in performance optimization and benchmarking



```python
from pydantic import BaseModel, Field, validator
import json
from google.protobuf.message import Message
from typing import Type, TypeVar, Generic, Any, Dict, Optional, Union

T = TypeVar('T', bound=BaseModel)
P = TypeVar('P', bound=Message)

class PydanticSerialization(Generic[T, P]):
    """Handles serialization between Pydantic models and wire formats."""

    def __init__(
        self,
        pydantic_model: Type[T],
        protobuf_message: Optional[Type[P]] = None,
        format: str = "json"
    ):
        self.pydantic_model = pydantic_model
        self.protobuf_message = protobuf_message
        self.format = format

    def serialize(self, data: T) -> bytes:
        """Serialize Pydantic model to bytes."""
        if self.format == "json":
            return data.json().encode("utf-8")
        elif self.format == "protobuf":
            if not self.protobuf_message:
                raise ValueError("Protobuf message type not provided")

            proto = self.protobuf_message()
            # Map Pydantic model to Protobuf message
            for field_name, field_value in data.dict().items():
                if hasattr(proto, field_name):
                    setattr(proto, field_name, field_value)
            return proto.SerializeToString()
        else:
            raise ValueError(f"Unsupported format: {self.format}")

    def deserialize(self, data: bytes) -> T:
        """Deserialize bytes to Pydantic model."""
        if self.format == "json":
            return self.pydantic_model.parse_raw(data.decode("utf-8"))
        elif self.format == "protobuf":
            if not self.protobuf_message:
                raise ValueError("Protobuf message type not provided")

            proto = self.protobuf_message()
            proto.ParseFromString(data)

            # Convert Protobuf message to dict for Pydantic
            result = {}
            for field in proto.DESCRIPTOR.fields:
                field_name = field.name
                if hasattr(proto, field_name):
                    result[field_name] = getattr(proto, field_name)

            return self.pydantic_model(**result)
        else:
            raise ValueError(f"Unsupported format: {self.format}")
```

Implementation Plan

Phase 1: Research & Foundation (Weeks 1-2)

- Set up development environment and analyze Dubbo-Python codebase
- Create prototype of Pydantic-based JSON serialization
- Design the core `PydanticSerialization` class architecture
- Define integration points with existing Dubbo-Python components

Phase 2: JSON Implementation (Weeks 3-4)

- Implement JSON serialization/deserialization using Pydantic
- Develop automatic validation during serialization/deserialization
- Create initial test suite for JSON serialization
- Begin integration with Dubbo-Python core components

Phase 3: Protobuf Integration (Weeks 5-6)

- Implement Protobuf support using `google.protobuf`
- Create mapping layer between Pydantic models and Protobuf messages
- Test interoperability with existing Protobuf implementations
- Benchmark performance against manual serialization

Phase 4: Validation & Optimization (Weeks 7-8)

- Enhance validation capabilities with custom validators
- Implement caching mechanisms for improved performance
- Optimize serialization for large payloads and high throughput
- Document best practices for model definition

Phase 5: Framework Integration (Weeks 9-10)

- Fully integrate with `Client`/`Server` classes
- Ensure backward compatibility with existing code
- Implement configuration options for serialization formats
- Create comprehensive integration tests

Phase 6: Testing & Documentation (Weeks 11-12)

- Develop end-to-end tests with client/server communication
- Test edge cases (e.g., invalid data, version mismatches)
- Create comprehensive documentation and usage examples
- Prepare final PR with code, tests, and documentation

Project Impact

Benefits to Dubbo Community

- **Simplified Development**: Eliminates repetitive serialization code
- **Increased Reliability**: Automatic validation prevents data errors
- **Feature Parity**: Brings Dubbo-Python closer to Dubbo-Java capabilities
- **Standardization**: Establishes consistent serialization practices
- **Community Growth**: Makes Dubbo more attractive to Python developers

Stretch Goals

If time permits, I aim to implement additional enhancements:

- Support for MessagePack serialization
- Performance benchmarking against Dubbo-Java
- Enhanced schema versioning and compatibility

Commitment

- **Availability**: 30-35 hours/week during GSoC period (May-August 2025)
- **Communication**: Weekly updates via email/Slack with mentors
- **Collaboration**: Active engagement with Dubbo community for feedback and testing
- **Long-term**: Commitment to maintain the serialization layer post-GSoC

Conclusion

The proposed serialization layer will significantly enhance Dubbo-Python's usability, bringing it closer to feature parity with its Java counterpart. My technical background in Python, distributed systems, and open-source contribution makes me well-equipped to deliver this solution successfully. I'm excited to contribute to Apache Dubbo and help expand its adoption in the Python ecosystem.