🤝 Contributing to MCP Git-Ollama Server

Thank you for your interest in contributing! This project is more than just a tool—it's a showcase of modern functional programming and a learning resource for the Scala community.

🎯 What We're Looking For

🔥 High-Impact Contributions

- **Performance optimizations** using functional programming techniques
- **New AI model integrations** beyond Ollama
- **Advanced functional programming patterns** (Free monads, MTL, etc.)
- **IDE integrations** for VS Code, IntelliJ, Vim, Emacs
- **Documentation improvements** with real-world examples

🧠 Learning Opportunities

Perfect for developers wanting to learn:

- **Cats Effect** and pure functional programming
- **Tagless Final** pattern implementation
- **HTTP4s** for functional HTTP clients
- **Circe** for JSON handling
- **Model Context Protocol** integration

🚀 Quick Start for Contributors

1. Development Setup

```
# Clone and build
git clone https://github.com/zlovtnik/git-mcp-commit-message.git
cd git-mcp-commit-message
sbt compile

# Run tests
sbt test

# Start development server
sbt run
```

2. Project Structure

```
src/main/scala/com/example/mcpgit/
├── core/           # Business logic and processing
├── git/            # Git operations and change analysis
```

```
├── ollama/          # AI model integration
├── server/          # MCP protocol handling
├── config/          # Configuration management
└── FunctionalMain.scala  # Application entry point
```

3. Architecture Principles

- **Pure Functions**: No side effects, referential transparency
- **Tagless Final**: Service algebras for testability
- **Error Handling**: Use `EitherT` for composable error handling
- **Resource Safety**: All resources managed with `Resource[F, _]`
- **Type Safety**: Opaque types and validated construction

🎨 Contribution Areas

🔧 Core Features

- [ ] **Caching Layer**: Cache AI responses for similar diffs
- [ ] **Template System**: Configurable commit message templates
- [ ] **Multi-Model Support**: Compare outputs from different models
- [ ] **Batch Processing**: Optimize for large repositories
- [ ] **Streaming**: Handle massive repositories efficiently

🌐 Integrations

- [ ] **VS Code Extension**: Native MCP client integration
- [ ] **IntelliJ Plugin**: IDEA ecosystem support
- [ ] **GitHub Actions**: CI/CD workflow integration
- [ ] **Pre-commit Hooks**: Git hook integration
- [ ] **CLI Tool**: Standalone command-line interface

⚡ Performance

- [ ] **Memory Optimization**: Reduce heap usage for large repos
- [ ] **Parallel Processing**: Multi-model concurrent analysis
- [ ] **Native Image**: GraalVM native compilation
- [ ] **Benchmarking**: Performance testing framework
- [ ] **Profiling**: Memory and CPU profiling tools

📚 Documentation

- [ ] **Architecture Guide**: Deep dive into FP patterns
- [ ] **Tutorial Series**: Learning functional programming
- [ ] **API Documentation**: Comprehensive API docs
- [ ] **Video Tutorials**: Screencast explanations
- [ ] **Use Case Examples**: Real-world scenarios

🧪 Testing Guidelines

Functional Testing Patterns

```
// Test with Cats Effect testing utilities
class GitServiceSpec extends AsyncFreeSpec with AsyncIOSpec {
  "GitService" - {
    "should parse git status correctly" in {
      val gitService = GitService.impl[IO]
      gitService.getStatus("/path/to/repo").asserting { changes =>
        changes should have size 3
        changes.head.changeType shouldBe ChangeType.Modified
      }
    }
  }
}
```

Property-Based Testing

```
// Use ScalaCheck for property-based tests
"RepoPath validation" - {
  "should accept valid paths" in {
    forAll(validPathGen) { path =>
      RepoPath(path) shouldBe a[Right[_, _]]
    }
  }
}
```

📝 Code Style

Functional Programming Guidelines

- **Use `F[_]` abstractions** instead of concrete types like `IO`
- **Prefer `EitherT`** for error handling over exceptions
- **Use opaque types** for domain modeling
- **Leverage type classes** for polymorphic behavior
- **Keep functions pure** and side effects in `F[_]`

Scala Style

```
// ✅ Good: Tagless final with error handling
trait UserService[F[_]] {
  def getUser(id: UserId): F[Either[UserError, User]]
}

// ✅ Good: Opaque types for type safety
opaque type UserId = String
object UserId {
```

```
  def apply(value: String): Either[ValidationError, UserId] =
    if (value.nonEmpty) Right(value) else Left(ValidationError("Empty
user ID"))
}

// ✅ Good: Resource management
def createService[F[_]: Async]: Resource[F, UserService[F]] =
  HttpClient.resource[F].map(UserService.impl[F])
```

🎯 PR Guidelines

Before Submitting

- ☐ **Tests pass**: `sbt test`
- ☐ **Code compiles**: `sbt compile`
- ☐ **Assembly works**: `sbt assembly`
- ☐ **Manual testing**: Verify MCP integration
- ☐ **Documentation updated**: README, FEATURES, etc.

PR Description Template

```
## 🎯 What This PR Does
Brief description of the changes.

## 🧪 Testing
- [ ] Unit tests added/updated
- [ ] Integration tests pass
- [ ] Manual testing completed

## 📚 Documentation
- [ ] README updated if needed
- [ ] Code comments added
- [ ] API docs updated

## 🔄 Breaking Changes
List any breaking changes and migration path.
```

🏆 Recognition

Contributors Hall of Fame

We celebrate contributors in multiple ways:

- **README acknowledgments** for significant contributions
- **Social media shoutouts** on project updates
- **Conference talk mentions** when presenting the project
- **Learning resources** featuring contributor examples

Mentorship Opportunities

- **Code reviews** with detailed functional programming feedback
- **Pair programming** sessions for complex features
- **Architecture discussions** for design decisions
- **Conference speaking** opportunities to present work

🌟 Getting Started Ideas

🟢 Beginner Friendly

- **Add new change type detection** (copy, symlink, etc.)
- **Improve error messages** with better context
- **Add configuration validation** with clear errors
- **Write example integrations** for different MCP clients

🟡 Intermediate

- **Implement caching layer** using Cats Effect `Ref`
- **Add metrics collection** with functional composition
- **Create template system** with type-safe configuration
- **Build performance benchmarks** using functional testing

🔴 Advanced

- **Design streaming architecture** for massive repositories
- **Implement Free monad** for effect composition
- **Add distributed processing** with functional concurrency
- **Create native image** compilation with GraalVM

💬 Community

Getting Help

- **GitHub Discussions**: Ask questions and share ideas
- **Issues**: Report bugs or request features
- **Discord/Slack**: Real-time chat (links coming soon)
- **Code Reviews**: Learn from detailed feedback

Sharing Knowledge

- **Blog Posts**: Write about your contributions
- **Conference Talks**: Present functional programming learnings
- **Tutorials**: Create learning content
- **Videos**: Record implementation explanations

🎓 Learning Resources

Functional Programming

- Cats Effect Documentation
- Tagless Final Pattern
- HTTP4s Documentation
- Scala 3 Reference

Project-Specific

- Model Context Protocol Spec
- Ollama API Documentation
- Git Porcelain Commands

Ready to contribute? Pick an issue, fork the repo, and let's build the future of AI-powered development tools together! 🚀