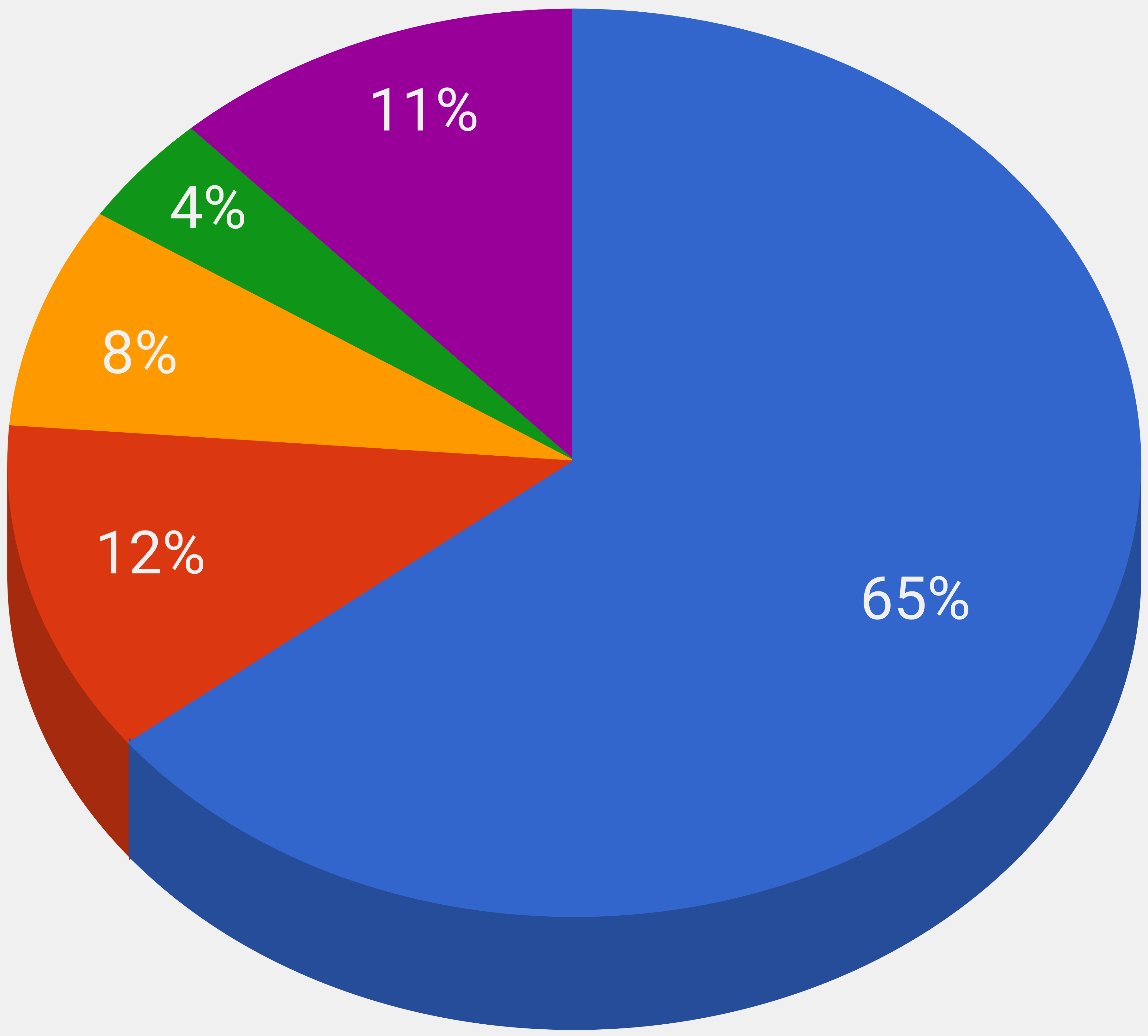
Different languages for AI applications

MARKET SHARE



● Python ● Typescript ● C# ● Go ● Others

Aleksei Kolesnikov
Staff Software Engineer

Language Ecosystem Analysis, In-depth comparison of C#, Go, Rust, Python, and JavaScript on AI Field

Aspect	C#	Go (Golang)	Rust	Python	JavaScript
AI Ecosystem	Limited: ML.NET minor DL support	Limited: Gorgonia, TF Go bindings	Limited: Nascent (tch-rs, burn)	Full: TensorFlow, PyTorch	Moderate (TensorFlow.js)
LLM Libraries	Full: SemanticKernel, langchain.net	Moderate: langchain-go	Minimal (langchain-rust)	Full: Autogen, langchain, crew.ai	Moderate: langchain.js, composio
Agentic Frameworks	Moderate: Autogen	Basic using go-ports and community libraries	Niche	Full: LangGraph, Crew.AI	Moderate: Composio
Document Analysis Capabilities	Moderate: Azure DocumentProcessing, Tesseract	Basic: go-fitz, go-pdf, unidoc	Limited (rust-pdf, lopdf)	Full: PyPDF2, spaCy, NLTK	Moderate: PDF.js, Instructor-js
Performance	Good (JIT compiled)	Excellent	Best-in-class for LLM Proxies	Moderate, GIL-limited	Moderate
Community & Enterprise Support	Moderate, Enterprise-focused	Growing (cloud-native emphasis)	Niche	Largest (academia/ industry)	Strong (web dev, limited AI)
Research Capabilities with Jupyter	Moderate: Jupyter with Interactive	Moderate: GoNB, GopherNotes	Niche	Full	Full: IJavascript

Aleksei Kolesnikov
Staff Software Engineer

Top Prompt Techniques per market

Market	Top Techniques	Language Alignment
Finance	Chain-of-Thought, Negative Example	C# (38%), Python (42%)
Healthcare	Role Prompting, Few-Shot Learning	Python (55%), JS (30%)
E-Commerce	Template Prompts, Dynamic Context	JS (48%), Go (22%)
Legal	Context Management, Role Priming	Python (60%), C# (25%)
Property Management	Chain-of-Thoughts, Zero-Shot	—

1. Iterative Prompt Versioning

Prompt history with A/B testing metrics (response accuracy, latency, cost).

2. Language-Specific Optimization

- C#,Go, TypeScript: Leverage strong typing for prompt template validation
- Python: Utilize Jupyter notebooks for rapid experimentation
- JS: Implement browser-based prompt preview systems

3. Cross-Platform Monitoring

- Token usage per language runtime
- Model-specific error patterns
- Context window utilization rates

Aleksei Kolesnikov
Staff Software Engineer

Quality Framework for LLM Applications

Phase	Key Components (Markdown Grid)	Verification Methods
Development	- Prompt validation pipelines - Context safety checks	Unit testing with adversarial examples
Testing	- Bias detection suites - Hallucination metrics	Differential testing across models
Deployment	- Real-time monitoring dashboards - Fallback mechanism triggers	Canary deployments with shadow traffic

This slide outlines a comprehensive quality assurance approach across different development phases:

> **Development:** Implement code & prompt validation pipelines and context safety checks to ensure AI behavior aligns with project goals. Unit Testing, General Prompt Testing, SonarQube.

> **Testing:** Employ bias detection suites, hallucination metrics, and ability to find prompt vulnerabilities. While traditional End-to-end automation and smoke testing are valuable, consider more dynamic testing strategies for rapidly evolving AI systems.

> **Deployment:** Utilize real-time monitoring dashboards, fallback mechanism, and models rotation.

Remember: Quality assurance in AI development requires continuous adaptation and vigilance across all phases.

Aleksei Kolesnikov
Staff Software Engineer

Key Focus Areas Grid

Field	Priority Prompt Tasks
Finance	Precision constraints, Regulatory checks
Healthcare	Role enforcement, Data anonymization
E-Commerce	Template consistency, Personalization
Cross-Platform	Context management, Error handling

Priorities vary across different sectors:

> **Finance:** Prioritize precision constraints and regulatory compliance checks to maintain accuracy and legal adherence.

> **Healthcare:** Focus on strict role enforcement and data anonymization to protect sensitive patient information.

> **E-commerce:** Emphasize template consistency and personalization to enhance user experience and drive conversions.

> **Cross-Platform:** Implement robust context management and error handling applicable across all markets.

Aleksei Kolesnikov
Staff Software Engineer

LLM Fine-tuning

Consider dedicated Python/ML specialists for LLM fine-tuning, if deemed necessary for your project goals

Vast majority for LoRA fine-tuning Libraries are in Python

- PETF
- Bitsandbytes (for quantization)
- HuggingFace Accelerate
- Hugging Face Transformers

Aleksei Kolesnikov
Staff Software Engineer

Executive Summary & Key Takeaway

While Python leads in ML development and fine-tuning, production-ready AI systems can be effectively built and deployed using various languages. Choose based on your team's skills, specific application needs, and performance requirements.

Important

1. Production Readiness:

- All major programming languages (Python, C#, Go, etc.) are capable of supporting AI and LLM applications in production.
- Startups and companies choose languages based on their team's expertise and specific use cases.

2. RAG Applications:

- For Retrieval-Augmented Generation (RAG) applications, using the team's preferred language is often the best choice.
- Python is not mandatory; prioritize the language your team is proficient in and can easily hire for.

3. Fine-Tuning Considerations:

- Fine-tuning LLMs may require a dedicated ML team and a separate budget.
- Carefully evaluate the necessity and goals of fine-tuning before committing resources.
- Python dominates in ML libraries and tools, making it the primary choice for fine-tuning tasks.

4. Production Performance:

- For high-throughput production environments, Go and C# may offer superior performance.
- These languages provide better options for fine-grained tuning and profiling in production systems.

Aleksei Kolesnikov
Staff Software Engineer