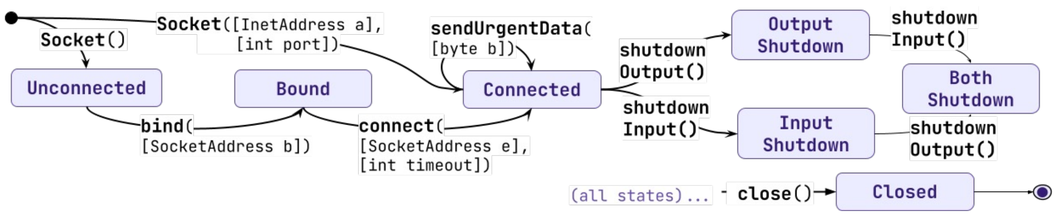
Barista: Synthesizing Typestate Specifications with LLM Agents

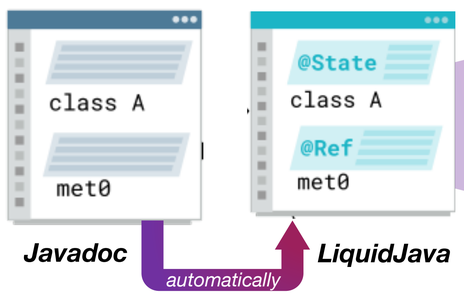
Catarina Gamboa[12], Paulo Canelas[12], Ricardo Costa[1],
Marcio Caetano[1], Jonathan Aldrich[2], Alcides Fonseca[1]

[1] LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
[2] S3D, Software and Societal Computing Department, Carnegie Mellon University

reliable software systems

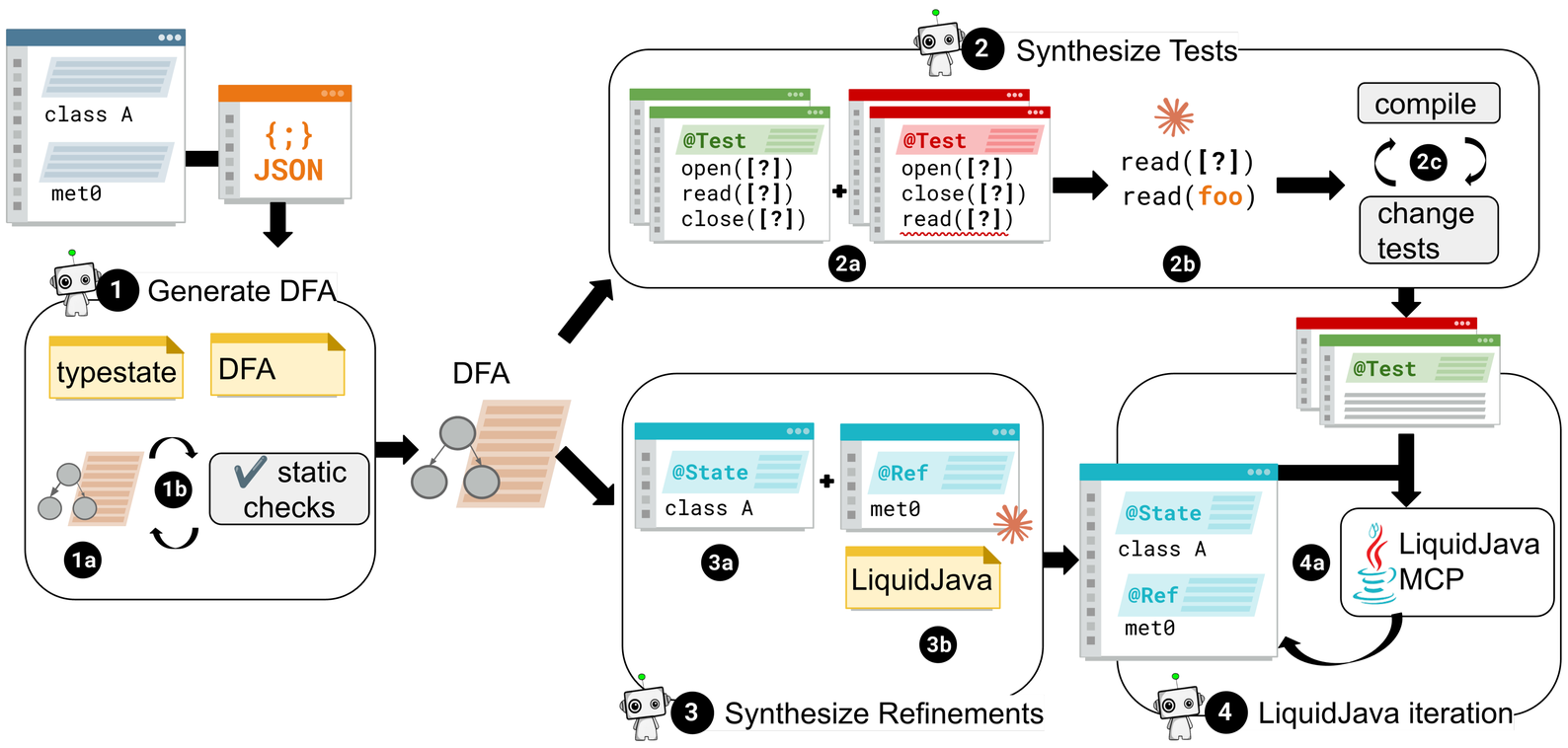
Object-oriented classes often encode implicit protocols that clients must follow, e.g `send()` is only possible if a socket was not already `close()`






```java
import liquidjava.specification.*;
import java.net.InetAddress;
import java.net.SocketAddress;

@StateSet({"unconnected", "bound", "connected", "inputshutdown"
      , "outputshutdown", "bothshutdown", "closed"})
@RefinementAlias("ValidPort(int p) {p >= 0 && p <= 65535}")
@ExternalRefinementsFor("java.net.Socket")
public interface SocketRefinements {
    // Constructors
    @StateRefinement(to = "unconnected(this)")
    public void Socket();

    @StateRefinement(to = "connected(this)")
    public void Socket(InetAddress address,
                       @Refinement("ValidPort(_)") int port);

    // Connection methods
    @StateRefinement(from = "unconnected(this)",
                     to = "bound(this)")
    public void bind(SocketAddress bindpoint);

    @StateRefinement(from = "bound(this)",
                     to = "connected(this)")
    public void connect(SocketAddress endpoint,
                        @Refinement("_ >= 0") int timeout);

    @StateRefinement(from = "connected(this)")
    public void sendUrgentData(int data);

    // Shutdown methods, shutdownOutput is similar
    @StateRefinement(from = "connected(this)",
                     to = "inputshutdown(this)")
    @StateRefinement(from = "outputshutdown(this)",
                     to = "bothshutdown(this)")
    public void shutdownInput();

    @StateRefinement(to = "closed(this)")
    public void close();
}
```

BARISTA: Agentic workflow to synthesize LiquidJava specifications from documentation



dataset

35	5	5-89
JDK Classes	Domains	Methods in classes
e.g., Socket, ImageReadParam, BufferedReader	*Graphics, I/O, System, Network, Security, Data*	

Are the generated specifications accurate? Why do they diverge from experts? Is every agent needed?

RQ1 How accurate are the generated specifications?

Can the generated specifications detect protocol violations?
Where do generated and expert specifications diverge, and why?

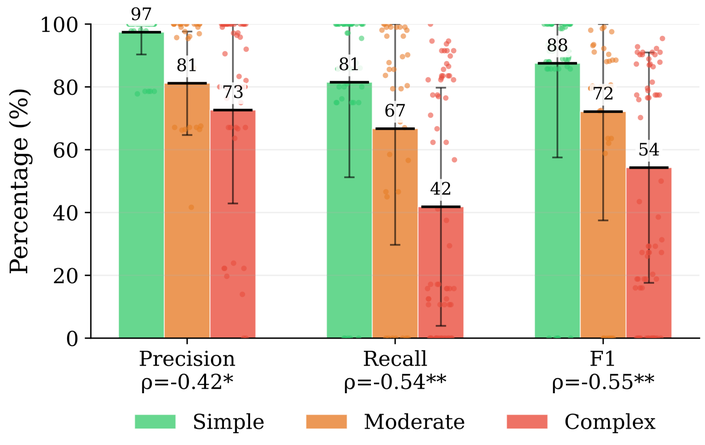
	Accuracy	Precision	Recall	F1
BARISTA	74.8%	87.0%	68.3%	76.5%

	Barista Detection		
	Always	*Partial*	*Never*
Expert ✅	23	5	1
Expert ❌	2	2	2

Failure root causes

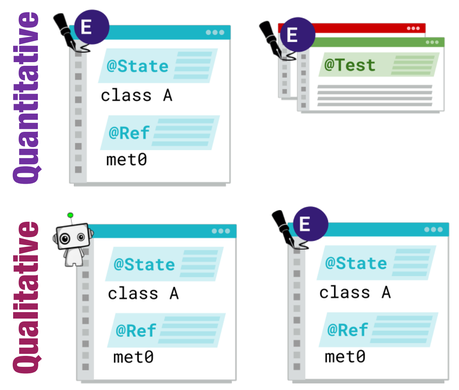
- 7 Parameter-conditional transitions
- 6 Getter under-restriction
- 5 Beyond-doc extrapolation
- 5 Ambiguous documentation
- 2 Orthogonal state dimensions
- 2 Source code dependency
- 2 Cross-class contract opacity

Information Gap / Modeling Gap



*Plot 1: Metric means and standard deviation per protocol complexity. Spearman rank correlations (ρ), * p<0.05, ** p<0.01.*

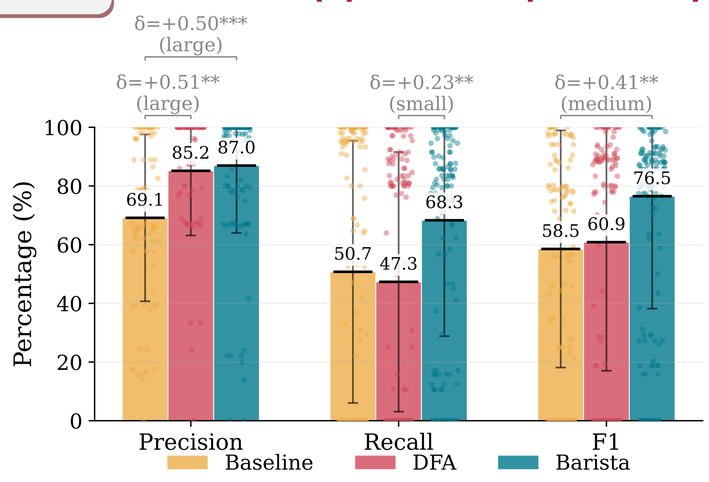
Expert Annotations



Answer RQ1

Barista achieves 87.0% precision and 68.3% recall. When divergences occur, Barista under-restricts rather than over-restricts, and 75% of specifications are usable directly or with minor edits.

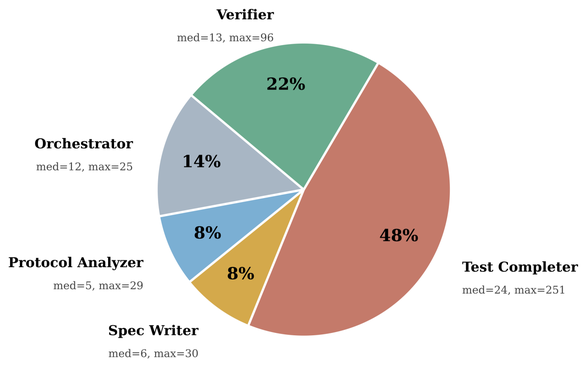
RQ2 Does each pipeline component improve specification quality?



*Plot 2: Ablation study. Brackets annotate significant improvements (Wilcoxon signed-rank, p<0.05) with Cliff's δ effect size; only non-negligible effects are shown. * p<0.05, ** p<0.01.*

Answer RQ2

DFA generation boosts precision; test-driven validation boosts recall. Barista: $2.20/class at 4.2× the speed of manual annotation.



Plot 3: Distribution of turns in each sub-agent

Discussion & Future Directions

1. Optimize for **precision over recall** to better support developer workflows with less false positives
2. More **expressive intermediate representations** can improve recall
3. **Better error messages** from the verifier can improve the refinement loop

Acknowledgements:
This work was funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the project CMU-Portugal, ref. PRT/BD/154254/2021 and SFRH/BD/151469/2021, and under the LASIGE Research Unit, ref. UID/00408/2025, DOI https://doi.org/10.54499/UID/00408/2025, which includes the LASIGE Seed project ref. LASIGE-SP-2025.07.
This work was also funded by the European Union - NextGenerationEU through the PRR - Recovery and Resilience Plan under the LASIGE Research Unit, ref. UID/PRR/00408/2025, DOI https://doi.org/10.54499/UID/PRR/00408/2025 and ref. UID/PRR2/00408/2025.