Industry Corner
Debdeep Paul

In this “Industry Corner” column, we interview Andrea Lops from Wideverse. Andrea Lops is a Ph.D. candidate in Electrical and Information Engineering at Politecnico di Bari, Italy, and a Machine Learning Engineer at Wideverse, where he has worked since 2019. He earned his M.Sc. in Computer Engineering from Politecnico di Bari with full marks and honors. His research sits at the intersection of software engineering and large language models (LLMs), with particular interests in AI-assisted and automated software testing, applied natural language processing, and knowledge engineering. He is the lead author of AgoneTest, a framework for automated unit-test generation and assessment with LLMs.

1. Could you tell us about your professional journey and how your interests in software engineering, machine learning, LLMs, and software testing developed over time?
My background is in software engineering. I started out as a mobile developer, building cross-platform and native apps at Nextome, and in 2019 I joined Wideverse, a spin-off of the Politecnico di Bari, where I worked across mobile development and machine learning (ML). That mix is really what shaped my interests: being close to products forced me to care about whether a model actually works in production, not just on a benchmark. Over time my focus shifted from conventional ML towards large language models (LLMs), and software testing became the thread that pulled everything together. It's the point where my engineering instinct and my interest in LLMs met, because you can use a model to generate code but you still have to prove that the code is correct. Starting my PhD at the Politecnico di Bari in 2023 was the natural next step: a way to take the problems I kept running into in industry and study them with proper rigor.

2. From your perspective, how have intelligent software systems - ranging from automation to LLM-assisted development and testing - evolved during your time in industry?
When I started, “intelligent” systems were mostly rule-based. I built early conversational agents with tools like Rasa and Mozilla DeepSpeech, and they worked, but every behavior had to be designed and scripted by hand. Capable LLMs changed the economics of that completely: suddenly you could get reasonable behavior out of the box and spend your effort on adaptation and integration instead of authoring rules. I've seen the same shift in testing, where work that used to be entirely manual, or limited to simple generators, is now something a model can draft for you, which is exactly the direction my own research took. The part that hasn't changed, and if anything has become harder, is deployment. In a small company you feel cost, latency, and reliability immediately, so a lot of the evolution I've lived through is less about raw model capability and more about learning how to wrap these models in something dependable enough to actually ship.

3. Your work touches multiple areas, from software engineering and machine learning to real-world deployment. How do you approach integrating ideas across these fields?
Honestly, I try not to treat machine learning and software engineering as separate worlds. I think of an LLM as one component in a larger system, a powerful and unpredictable one, but still a component with inputs, outputs, and failure modes like any other. Once you see it that way, the engineering discipline you'd apply to anything else applies here too: you specify what you expect, and you test it. My work on automated test generation is basically that idea taken to its conclusion. A model can write the tests, but I then need a way to assess whether those tests are any good, so the ML side produces the artifact and the software-engineering side keeps it honest. Integrating the fields, for me, is less a clever trick and more a default posture: use the model where it is strong, and put rigor around it where it is weak.

4. What do you see as the unique strengths and challenges of doing research within an industrial R&D or spin-off environment? What broad trends do you think will define the next decade of progress in AI-assisted software engineering?

The biggest strength of working in a spin-off like Wideverse is the short distance between a research idea and a real user. I have access to actual data and actual deadlines, which keeps the work honest, because a method that only looks good in a paper rarely survives contact with a product. Being attached to the Politecnico di Bari at the same time gives me the other half: the freedom, and the obligation, to step back and study a problem properly instead of just patching it. The challenge is the tension between those two time frames. Industry rewards shipping and research rewards rigor, and a spin-off lives right on that line, so you're constantly negotiating between "good enough to deploy" and "sound enough to publish." As for the next decade, I don't think the defining change will be a single bigger model, but the way these models get woven into the everyday development workflow, with coding and testing assistants acting more autonomously, and alongside that a far greater emphasis on evaluating and trusting what they produce. The teams that take evaluation seriously are the ones whose tools people will actually rely on.

5. As AI and LLMs become more embedded in software development, how should researchers think about responsibility, trust, reliability, and safety in the systems they design?

My honest view is that you shouldn't trust an LLM's output, you should verify it. That sounds blunt, but it's the healthiest default, and it's really the whole motivation behind my research: if a model writes code or tests for you, you need an independent way to check that what it produced is actually correct. So reliability has to be something you measure. We should hold these systems to reproducible evaluation rather than to demos and anecdotes, because a system that impresses in a screenshot can fail quietly in production. And underneath all of that there has to be a human who stays accountable, because the model is an assistant and the engineer is the one who signs off. A lot of the responsibility conversation gets clearer once you stop asking whether a model is trustworthy in the abstract and start asking how you'd measure its reliability and who is answerable when it's wrong.

6. For students and early-career professionals interested in AI for software engineering, LLMs, and intelligent systems, what foundational skills or mindsets do you consider essential?

My main advice is to resist the temptation to start from the LLM. The fundamentals matter more than ever, algorithms, data structures, software engineering, the basics of machine learning, because they are what let you tell whether an impressive-looking output is actually correct. I spend a fair amount of time teaching algorithms, and I'm convinced the people who reason well about a problem get the most out of these tools, not the ones who can write the cleverest prompt. On top of that you need genuine critical evaluation: treat AI output as a hypothesis to be checked, not an answer to be trusted. And you have to accept that the field moves fast, so the real skill is learning how to learn, reading papers and trying things and updating quickly. Strong foundations, a habit of scepticism, and the ability to keep up: that combination ages far better than any specific tool you could master today.

7. Beyond technical expertise, what non-technical skills have been most helpful to you, and what advice would you give to young researchers who hope to bridge strong academic foundations with impactful real-world applications?

The skill I've come to value most is problem framing. When tools and models are abundant, the bottleneck isn't building something, it's deciding what's actually worth building and turning a vague need into a question you can solve and measure. Close behind that is collaboration. Almost everything I've worked on sits between people (engineers, researchers, and domain experts who understand the problem far better than I do) and the good ideas tend to show up at those intersections rather than alone at a desk. My advice to younger researchers is simple: keep your curiosity and your standards high, but always keep one foot in the real world. Academic depth is what makes your work last, and real-world contact is what makes it matter, and if you can hold both at once your research stops being an exercise and starts being something people use.