Agent-as-a-Judge Presentation Speaker Notes

30-Second Speech for Each Slide

Based on ICML 2025 Paper
Meta AI & KAUST

November 14, 2025

Instructions

Each slide has a 30-second speech prepared. Practice these to maintain consistent timing. Total presentation time: approximately 10-11 minutes. Speak clearly and maintain eye contact with the audience. Pause after key points for emphasis.

Contents

1 Slide 1: Title Slide

Visual: Title, Authors, Institutions

Good morning everyone. Today I'm excited to present Agent-as-a-Judge, a groundbreaking framework for evaluating agentic systems using agents themselves. This is joint work between Meta AI and KAUST that will appear at ICML 2025. The key insight is simple but powerful: if agentic systems can solve complex multi-step problems, they should also be able to evaluate other agentic systems. Over the next ten minutes, I'll show you how we've made this vision a reality with a framework that's as reliable as human evaluators but ninety-seven percent cheaper.

Key Points to Emphasize

- Paper accepted to ICML 2025
- Joint work: Meta AI and KAUST
- Main contribution: using agents to evaluate agents

2 Slide 2: Agenda

Visual: Four-part outline of presentation

Our presentation has four parts. First, I'll explain the problem: why current evaluation methods fail for agentic systems. Second, I'll introduce our solution: the DevAI dataset with fifty-five realistic AI development tasks. Third, I'll dive into the Agent-as-a-Judge framework itself, showing how it works and why it's effective. Finally, I'll share our results demonstrating ninety percent alignment with human judges at a fraction of the cost. Let's get started.

Key Points to Emphasize

- Four clear parts: Problem, Dataset, Framework, Results
- Sets expectations for audience
- Approximately 10 minutes total

3 Slide 3: Motivation

Visual: Problem statement with examples

Agentic systems are rapidly moving from toy problems to real-world deployments. They think step-by-step, have internal reasoning processes, and solve complex tasks autonomously. But here's the problem: our evaluation methods haven't kept up. We're still evaluating these sophisticated systems like we would a simple function—just checking if the final output is correct. This is like judging a student's understanding using only multiple-choice tests. We're missing all the intermediate reasoning, the problem-solving approach, and the actual thinking process. We need evaluation methods that match the sophistication of what we're evaluating.

Key Points to Emphasize

- Agentic systems are being deployed in production
- Current evaluation only looks at final outcomes
- Missing intermediate reasoning and process
- Analogy: multiple-choice tests vs. comprehensive evaluation

4 Slide 4: Current Benchmarks Fall Short

Visual: Comparison of existing benchmarks

Let's look at existing benchmarks. HumanEval tests only algorithmic problems. MBPP covers simple programming tasks. SWE-Bench, while more realistic, only looks at final resolve rates without explaining what went wrong. Worse, recent research shows that twenty-seven percent of SWE-Bench tasks can be solved by LLMs alone without any agentic features. We're also seeing Goodhart's Law in action: when a measure becomes a target, it ceases to be a good measure. Teams are optimizing specifically for these benchmarks rather than building truly capable systems. We need benchmarks that reflect real development workflows and provide rich interpretable feedback.

Key Points to Emphasize

- Existing benchmarks too simple or narrow
- Binary success metrics insufficient
- Goodhart's Law: optimization kills validity
- Need: realistic tasks with rich feedback

5 Slide 5: From LLM-as-a-Judge to Agent-as-a-Judge

Visual: Comparison diagram

This brings us to evaluation methods. LLM-as-a-Judge has been useful for evaluating conversational tasks—it's a single LLM call that compares outputs. But it has critical limitations: no tool use, no environment interaction, and most importantly, no ability to provide intermediate feedback. Agent-as-a-Judge is our natural evolution of this concept. It incorporates agentic features: multi-step reasoning, tool use, environment interaction, and the ability to evaluate throughout the entire process. Think of it as upgrading from a static judge to one who can actively investigate and verify claims.

Key Points to Emphasize

- LLM-as-a-Judge: single call, limited context
- Agent-as-a-Judge: tools, reasoning, interaction
- Key difference: intermediate feedback capability

6 Slide 6: Main Contributions

Visual: Four-step development process

Our work makes four key contributions. First, we created DevAI: a dataset of fifty-five realistic AI development tasks with three hundred sixty-five hierarchical requirements. Second, we benchmarked three leading open-source frameworks—MetaGPT, GPT-Pilot, and OpenHands—with comprehensive human evaluation taking eighty-six hours. Third, we introduced the Agent-as-a-Judge framework with five core components that work together to evaluate code-generating agents. Finally, we validated our approach, achieving ninety percent alignment with human consensus while reducing evaluation cost by ninety-seven percent. These four steps demonstrate a complete solution to the evaluation problem.

Key Points to Emphasize

- DevAI: 55 tasks, 365 requirements
- Human baseline: 86.5 hours of evaluation
- Agent-as-a-Judge: 5 modular components
- Results: 90% alignment, 97% cost reduction

7 Slide 7: DevAI Dataset Overview

Visual: Dataset statistics and design principles

Let me tell you about DevAI. We chose AI development tasks because they follow clear, standard procedures with natural topological structure: data processing, model implementation, training, evaluation, and reporting. This structure helps us monitor the development process and provide useful feedback signals. The dataset contains fifty-five carefully curated tasks covering supervised learning, computer vision, NLP, reinforcement learning, and more. Each task is small-scale but realistic, computationally inexpensive to run, yet rich in hierarchical requirements. This design prevents simple memorization while enabling thorough evaluation of the complete development cycle.

Key Points to Emphasize

- Why AI tasks: clear procedures, topological structure
- 55 tasks across multiple AI domains
- Small-scale but realistic
- Prevents memorization, enables rich evaluation

8 Slide 8: DevAI Structure Details

Visual: Example task with hierarchical requirements

Here's how DevAI works in practice. Each task starts with a user query—for example, "generate ten-eighty-p images with hidden text." Then we have hierarchical requirements structured as a directed acyclic graph. R-zero depends on nothing: follow the blog instructions. R-one depends on R-zero: generate ten-eighty-p images. R-two depends on R-one: embed the specific text. This creates a clear dependency chain. Requirements are explicit and binary—either satisfied or not—making evaluation unambiguous. We also include optional preferences for softer criteria. This structure provides rich feedback: you know exactly which steps succeeded and which failed.

Key Points to Emphasize

- User query + hierarchical requirements
- DAG structure with dependencies
- Requirements: explicit and binary
- Rich feedback: know exactly what failed

9 Slide 9: Baseline Agent Performance

Visual: Performance comparison table

We evaluated three leading frameworks, all using GPT-4o as the backend. MetaGPT is the most cost-efficient at one-nineteen USD per task but generates fewer files. GPT-Pilot takes the longest time but produces the most code output. OpenHands is fastest and has the best user experience but is most expensive at six-thirty-eight per task. Looking at actual performance: the best agents only satisfy about twenty-nine percent of requirements and fully solve just one task out of fifty-five. This shows DevAI provides an appropriate level of challenge for current systems—it's hard but not impossible.

Key Points to Emphasize

- Three agents: MetaGPT, GPT-Pilot, OpenHands
- Trade-offs: cost vs. time vs. output
- Best performance: 29% requirements, 1 full task
- Appropriate difficulty level

10 Slide 10: Human Evaluation Process

Visual: Two-round evaluation process

To establish a reliable baseline, we conducted rigorous human evaluation in two rounds. Round one: three expert evaluators with over five years of AI research experience each spent nearly thirty hours doing individual evaluations with minimal instructions—this captures natural human bias. Round two: twenty-eight hours of consensus building where evaluators presented evidence, debated disagreements, and reached agreement on each requirement. This process corrects individual errors through peer review and reduces cognitive biases. The total cost was one hundred fifteen human hours or about thirteen hundred dollars. This consensus evaluation became our gold standard.

Key Points to Emphasize

- Round 1: Independent evaluation (86.5 hours total)
- Round 2: Consensus building (28.5 hours)
- Three expert evaluators, 5+ years experience
- Total: 115 hours, $1,300 cost

11 Slide 11: Human Evaluation Reliability

Visual: Disagreement rates and error analysis

But here's something important: human evaluators disagree—a lot. Disagreement rates ranged from ten to twenty-seven percent between pairs of evaluators. Why? These are complex multi-step tasks where it's easy to miss critical information or interpret ambiguity differently. The key insight: majority voting dramatically reduces errors. Individual evaluators had error rates of ten to twenty-four percent compared to consensus, but majority voting reduced this to just five to six percent. We validated this with ten additional experts who showed ninety-five percent agreement with our consensus. This demonstrates the inherent challenge of evaluation and the value of ensemble methods.

Key Points to Emphasize

- Individual disagreement: 10-27%
- Individual errors: 10-24% vs. consensus
- Majority voting: only 5-6% error
- Validation: 10 experts, 95% agreement

12 Slide 12: Agent-as-a-Judge Design

Visual: System architecture diagram

Now let's dive into Agent-as-a-Judge itself. We designed it to imitate how humans evaluate: navigate codebases systematically, read files, check outputs, and verify requirements. We initially designed eight modular components but through ablation studies found that five are essential. Graph builds the workspace structure. Locate finds relevant files. Read extracts contents from thirty-three different file formats. Retrieve checks execution logs and trajectories. Ask makes the final judgment with justification. The components we removed—Search, Planning, and Memory—either added noise or created unstable behavior. This modular design is both effective and interpretable.

Key Points to Emphasize

- Philosophy: imitate human evaluation process
- Initial: 8 components; Final: 5 essential ones
- Core 5: Graph, Locate, Read, Retrieve, Ask
- Removed: Search (noise), Planning (unstable), Memory (error chains)

13 Slide 13: Component Details

Visual: Detailed component descriptions

Let me explain each component's contribution. Graph adds nearly eleven percent by understanding file relationships. Read adds six percent by accessing multimodal content. But Locate is the star, adding eight percent by intelligently focusing on relevant files—this is crucial because it prevents information overload. Retrieve is valuable in gray-box settings where we have trajectory data, particularly for agents like GPT-Pilot. Ask synthesizes all evidence to make the final judgment with clear reasoning. The sequential performance gains show that each component builds on the previous ones, creating a comprehensive evaluation system.

Key Points to Emphasize

- Graph: +11% (relationships)
- Read: +6% (multimodal access)
- Locate: +8% (intelligent focus)
- Retrieve: variable (trajectory data)
- Ask: synthesis and judgment

14 Slide 14: Evaluation Results

Visual: Alignment rate comparison table

Here are our main results. In black-box settings without trajectory access, Agent-as-a-Judge achieves eighty-eight to ninety percent alignment with human consensus—dramatically outperforming LLM-as-a-Judge's sixty to eighty-four percent. With trajectory data in gray-box settings, Agent-as-a-Judge reaches ninety-two percent alignment. Compare this to individual human evaluators at eighty-five to ninety percent. Our framework sometimes exceeds individual human performance while approaching the gold standard of human majority vote at ninety-four percent. This establishes a new reliability ranking: human majority vote, then Agent-as-a-Judge, then individual humans, and finally LLM-as-a-Judge.

Key Points to Emphasize

- Black-box: 88-90% alignment
- Gray-box: 92% alignment
- Outperforms LLM-as-a-Judge (60-84%)
- Comparable to individual humans (85-90%)
- Near human majority vote (94%)

15 Slide 15: Judge Shift Analysis

Visual: Deviation metrics

Judge Shift measures deviation from human consensus—lower is better. Agent-as-a-Judge shows remarkable stability with shifts as low as zero-point-two-seven percent for OpenHands. LLM-as-a-Judge shows much larger shifts, particularly for complex agents like GPT-Pilot at thirty-two percent. Why does Agent-as-a-Judge win? Three reasons: intelligent context selection that focuses on relevant evidence rather than overwhelming with all data, multi-step reasoning that builds understanding incrementally, and tool use that enables active workspace exploration and empirical verification. These agentic features make all the difference.

Key Points to Emphasize

- Judge Shift: measures deviation (lower better)
- Agent-as-a-Judge: 0.27-8.20%
- LLM-as-a-Judge: up to 32%
- Why it wins: context selection, reasoning, tools

16 Slide 16: Ablation Studies

Visual: Component contribution analysis

Our ablation studies reveal interesting insights. Starting with just Ask at sixty-five percent, adding components sequentially brings us to ninety percent. But not everything helps: adding Search actually decreased performance because BM-twenty-five retrieval introduced noise, and our workspaces were too simple to benefit from code search. Planning was unstable, creating inconsistent evidence. Memory created error chains where mistakes in earlier judgments biased later ones. For trajectory retrieval, we found that keeping the final sections is crucial—that's where the dense information about final state resides. These findings guide future improvements.

Key Points to Emphasize

- Sequential gains: 65% → 90%
- Search decreased performance (noise)
- Planning unstable, Memory creates error chains
- Keep final trajectory sections (dense info)

17 Slide 17: Cost and Efficiency Analysis

Visual: Cost comparison table

Now for the practical impact. Human evaluation costs thirteen hundred dollars and takes one hundred fifteen hours. LLM-as-a-Judge is fast at eleven minutes but costs thirty dollars. Agent-as-a-Judge takes one-eighteen minutes and costs thirty-one dollars—achieving ninety-seven-point-seven percent time reduction and ninety-seven-point-six percent cost reduction compared to humans. This is transformative. Imagine testing ten agent variations: that's eleven hundred hours and thirteen thousand dollars for humans versus twenty hours and three hundred dollars for Agent-as-a-Judge. This makes rigorous iterative development actually practical. Small labs can now afford comprehensive evaluation.

Key Points to Emphasize

- Human: 115 hours, $1,300
- Agent-as-a-Judge: 2 hours, $31
- Savings: 97.7% time, 97.6% cost
- Enables rapid iteration
- Democratizes access to rigorous evaluation

18 Slide 18: Failure Analysis

Visual: Failure cases by category

Let's be honest about limitations. Agent-as-a-Judge struggles most with data preprocessing—ten failures—and dataset identification—eight failures. For example, it was fooled by synthetic datasets that mimicked real ones in naming. It also missed nuances like distinguishing between setting hyperparameters and actually tuning them dynamically. The system performs best on clear, verifiable outputs like HCI and visualization. These failure cases point to future improvements: better data provenance verification, execution-based validation, and deeper semantic analysis. We need domain-specific checks for subtle requirements.

Key Points to Emphasize

- Most failures: data preprocessing (10), datasets (8)
- Example: fooled by synthetic data naming
- Example: missed "tuning" vs. "setting" nuance
- Best: HCI, visualization (clear outputs)
- Future: provenance verification, execution validation

19 Slide 19: Impact and Future Directions

Visual: Applications and extensions

The impact is immediate and far-reaching. Researchers get affordable rigorous evaluation with rich interpretable feedback. Practitioners can integrate this into continuous testing and quality assurance pipelines. But the most exciting opportunity is interactive feedback loops: imagine Agent-as-a-Judge providing feedback during development, the developer agent using that feedback to fix issues in real-time, creating a mutual improvement cycle. Beyond code generation, the modular design enables easy adaptation to web agents, robotics, and other domains. We're also working on automated prompt optimization and workflow learning. This framework opens the door to vastly more sophisticated agentic systems.

Key Points to Emphasize

- Immediate: affordable evaluation for researchers
- Practitioners: CI/CD integration
- Future: interactive feedback loops
- Extensions: web agents, robotics, other domains
- Optimization: automated prompt/workflow learning

20 Slide 20: Conclusion

Visual: Key takeaways

Let me summarize. The problem is real: current evaluation methods are inadequate for agentic systems. We provided two solutions: DevAI with fifty-five realistic tasks and Agent-as-a-Judge framework. It works: ninety percent alignment with humans, ninety-seven percent cost reduction, outperforming LLM-as-a-Judge while being comparable to expert evaluators. This is a concrete step toward enabling vastly more sophisticated agentic systems. Agent-as-a-Judge is a natural extension of LLM-as-a-Judge that incorporates agentic features to provide intermediate feedback while dramatically reducing cost. The dataset and code are publicly available. Thank you, and I'm happy to take questions.

Key Points to Emphasize

- Problem: inadequate evaluation for agentic systems
- Solution: DevAI dataset + Agent-as-a-Judge
- Results: 90% alignment, 97% cost reduction
- Available: github.com/metauto-ai/agent-as-a-judge

21 Slide 21: Questions and Discussion

Visual: Contact information

I'm now happy to take your questions. We have time for discussion about extending this to other domains, scaling to longer tasks, handling adversarial cases, or integration with existing systems. You can reach us at the emails shown, and remember that all code and data are available on GitHub. Thank you for your attention, and I look forward to your questions.

Key Points to Emphasize

- Open floor for questions
- Topics: extensions, scalability, integration
- Emphasize open-source availability
- Contact: mingchen.zhuge@kaust.edu.sa, cszhao@meta.com

Presentation Tips

Timing

- Total time: 10-11 minutes
- 30 seconds per slide average
- Build in 5-10 seconds buffer between sections
- Practice with a timer

Delivery

- Maintain eye contact with audience
- Pause after key statistics for emphasis
- Use hand gestures for "three reasons," "four contributions," etc.
- Slow down for technical terms (e.g., "directed acyclic graph")
- Show enthusiasm when discussing results

Key Numbers to Remember

- 55 tasks, 365 requirements, 125 preferences
- 3 agents evaluated: MetaGPT, GPT-Pilot, OpenHands
- 90% alignment with human consensus
- 97.7% time reduction, 97.6% cost reduction
- 5 core components (Graph, Locate, Read, Retrieve, Ask)

Anticipated Questions

1. **Q: Can this work for domains beyond code generation?**
   A: Yes, the modular design makes it easily adaptable. We're exploring web agents and robotics applications.

2. **Q: What happens when Agent-as-a-Judge disagrees with humans?**
   A: We analyzed failure cases—mostly in data preprocessing and dataset identification. Section 18 covers this.

3. **Q: How do you prevent the agent from gaming the evaluation?**
   A: Good question. The hierarchical requirements with dependencies make gaming difficult, and we evaluate processes not just outcomes.

4. **Q: What's the computational cost during evaluation?**
   A: About $30 per full evaluation run—dramatically cheaper than human evaluation while maintaining reliability.

5. **Q: Can smaller models work as judges?**
   A: We tested Claude and found it performs even better. Table 10 in our paper shows ablations with different LLM backbones.