



When OR

meets

Generative AI

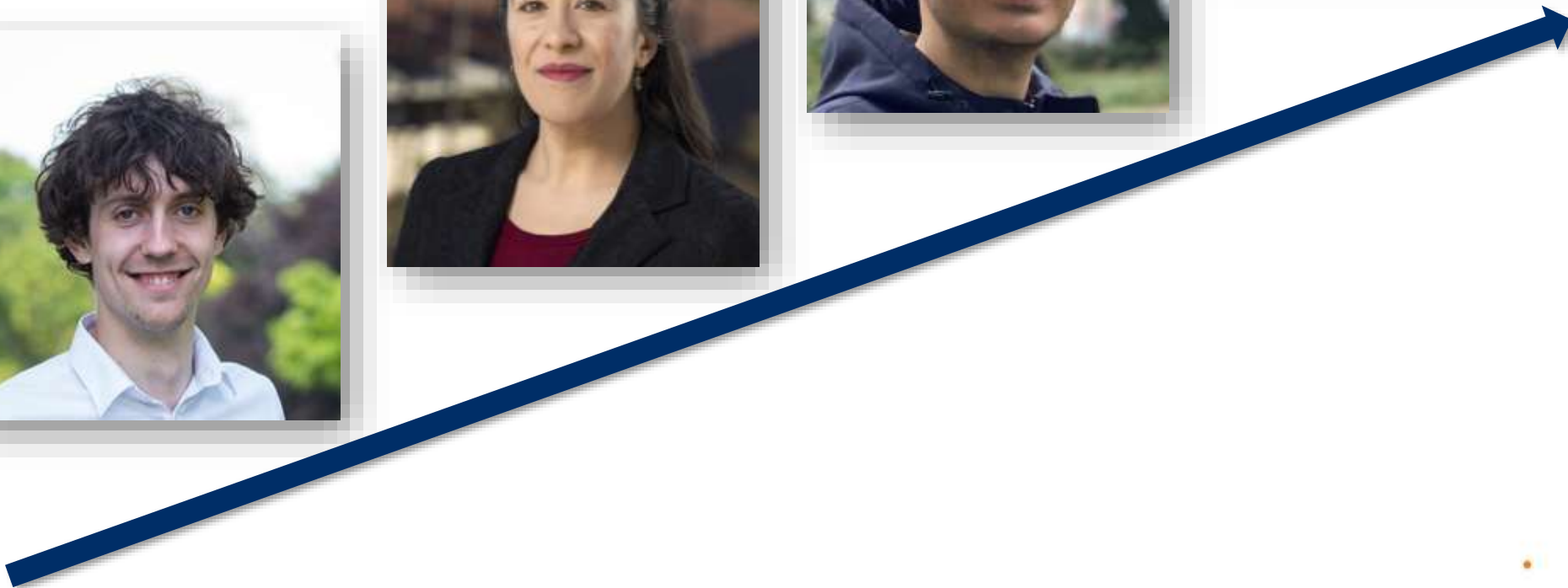
Wouter Kool

ORTC

The lineup



The line up



The warmup: what & when?



ORTEC

AlphaGo beats Lee Sedol (2016)



The Adam optimizer (Kingma & Ba, 2015)

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

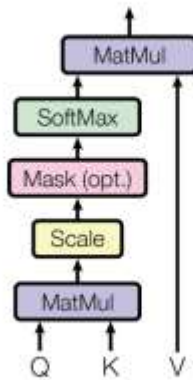
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

end while

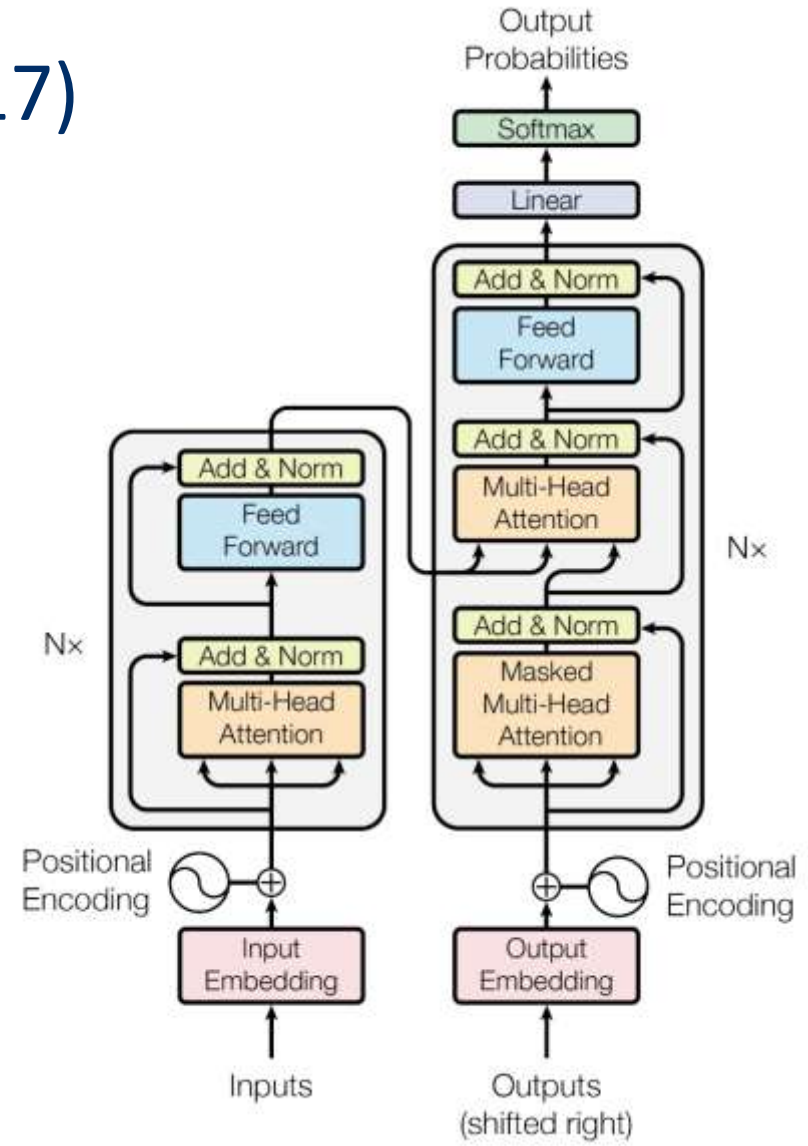
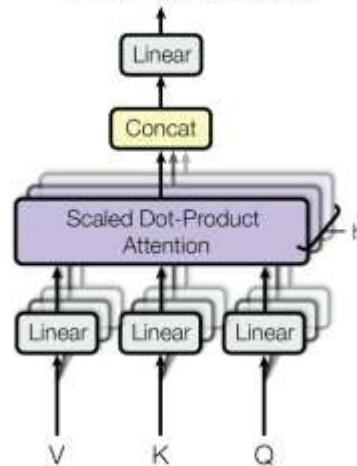
return θ_t (Resulting parameters)

The Transformer (Vaswani et al., 2017)

Scaled Dot-Product Attention



Multi-Head Attention



ChatGPT released (2022)

ChatGPT



Examples

"Explain quantum computing in simple terms"

"Got any creative ideas for a 10 year old's birthday?"

"How do I make an HTTP request in Javascript?"



Capabilities

Remembers what user said earlier in the conversation

Allows user to provide follow-up corrections

Trained to decline inappropriate requests



Limitations

May occasionally generate incorrect information

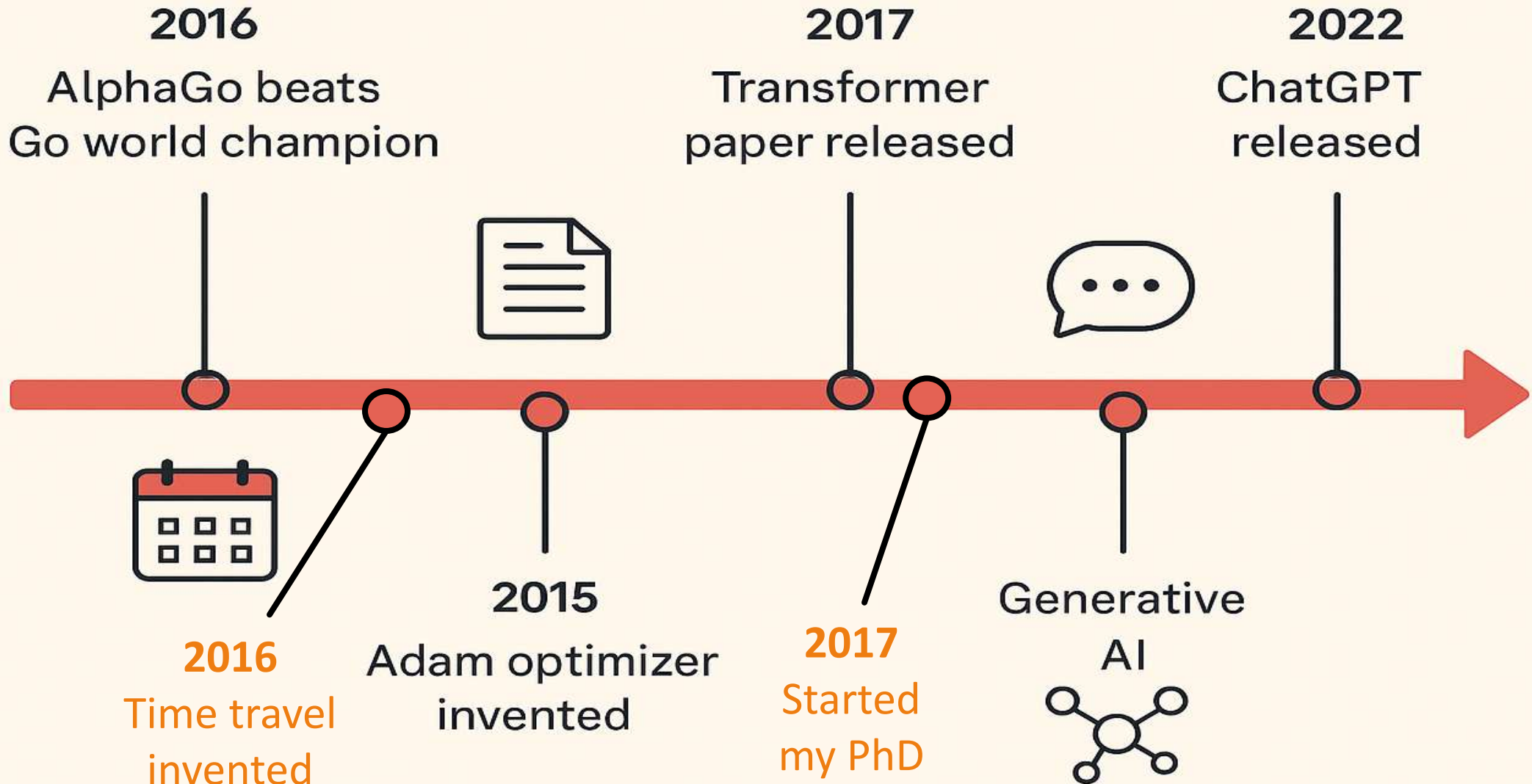
May occasionally produce harmful instructions or biased content

Limited knowledge of world and events after 2021



Free Research Preview: ChatGPT is optimized for dialogue. Our goal is to make AI systems more natural to interact with, and your feedback will help us improve our systems and make them s

AI TIMELINE (2016-2025)

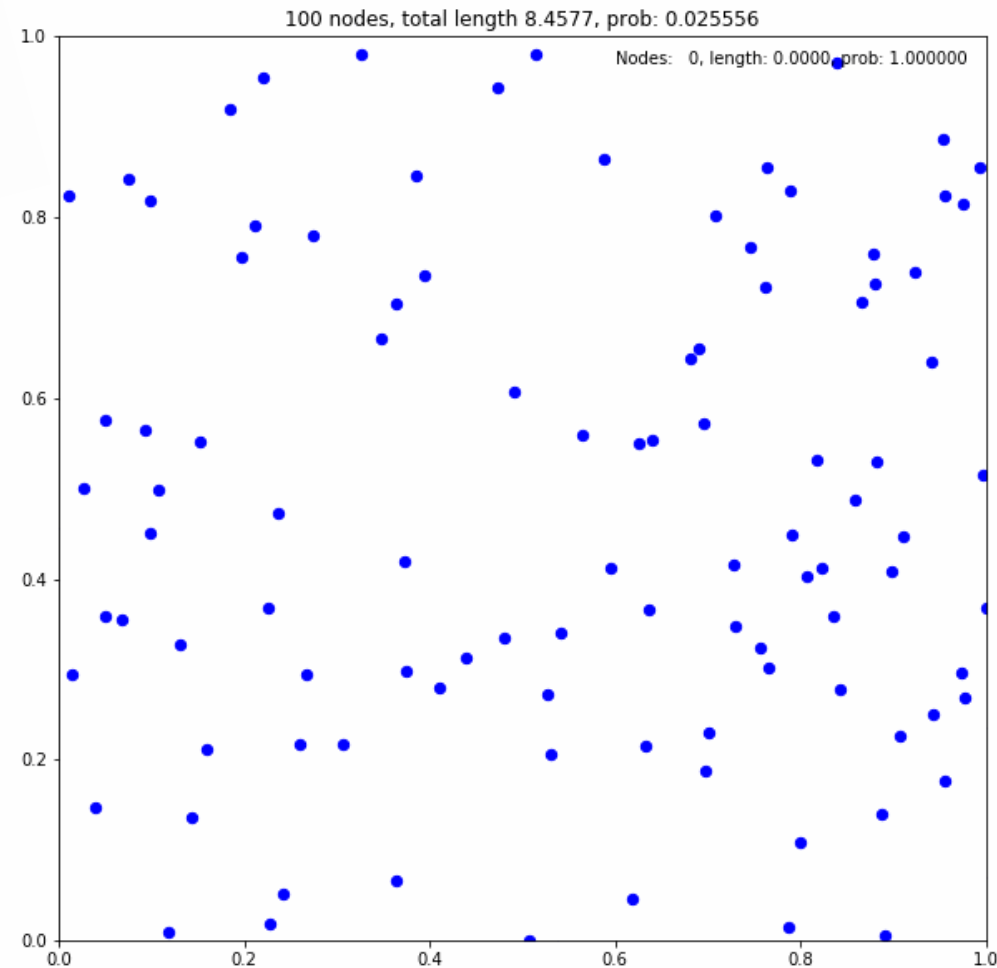
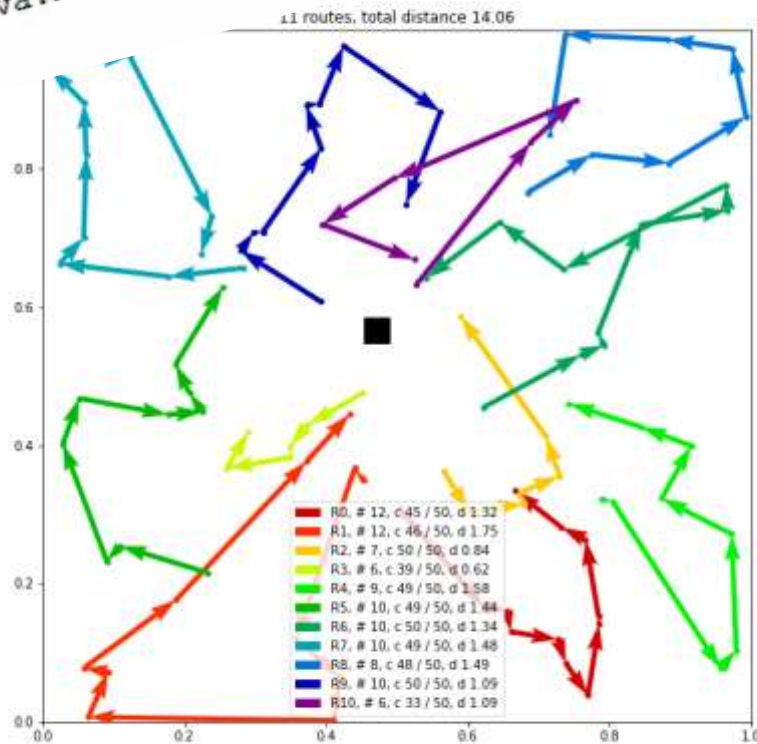


ATTENTION, LEARN TO SOLVE ROUTING PROBLEMS!

Wouter Kool
University of Amsterdam
ORTEC
w.w.m.kool@uva.nl

Herke van Hoof
University of Amsterdam
h.c.vanhoof@uva.nl

Max Welling
University of Amsterdam
CIFAR
m.welling@uva.nl



Real world problems...

Raeal



Real world problems are rarely as clean as our equations!

- A wise LLM that wrote my abstract -



5 years from now?

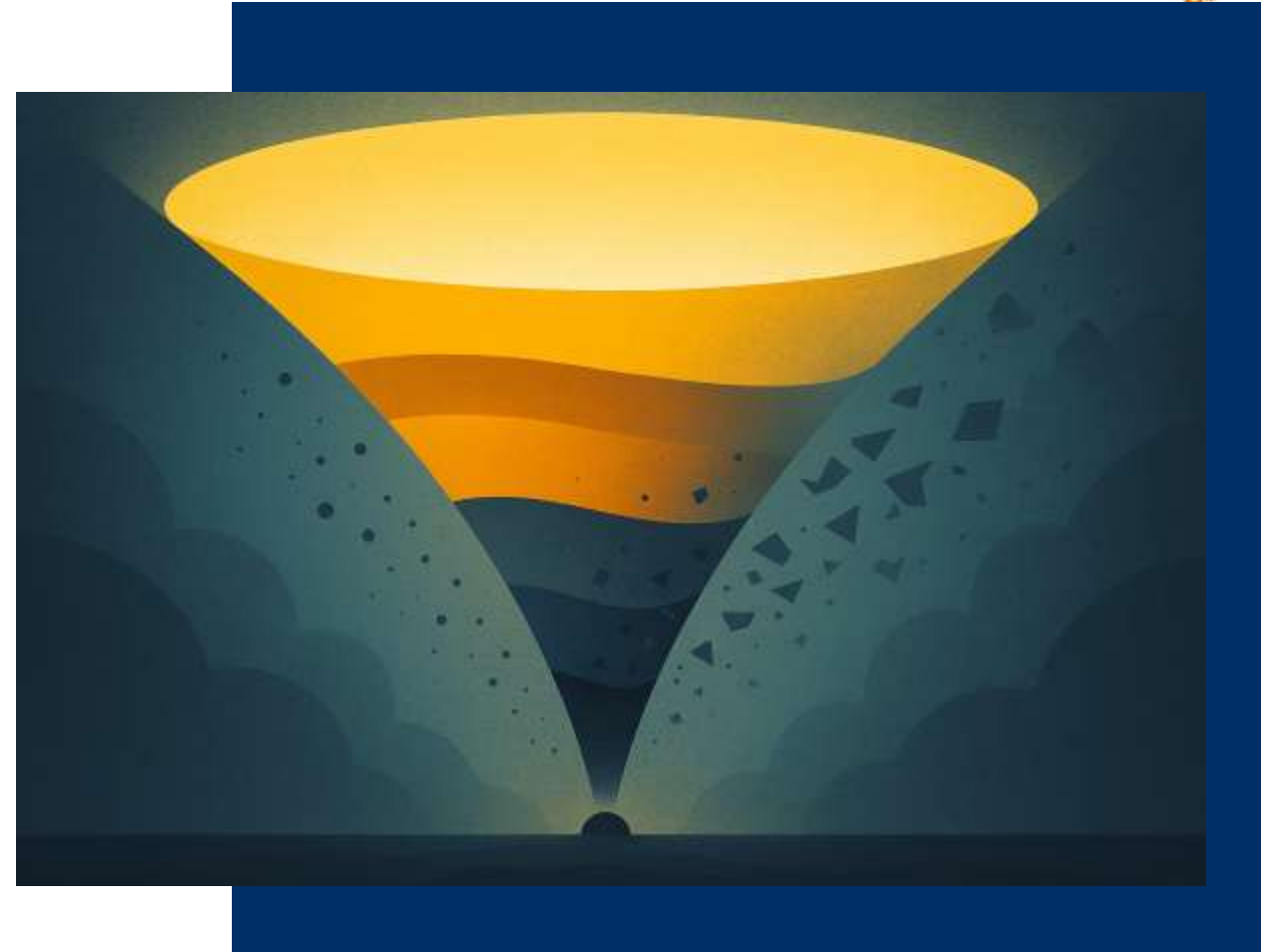


In a typical OR application

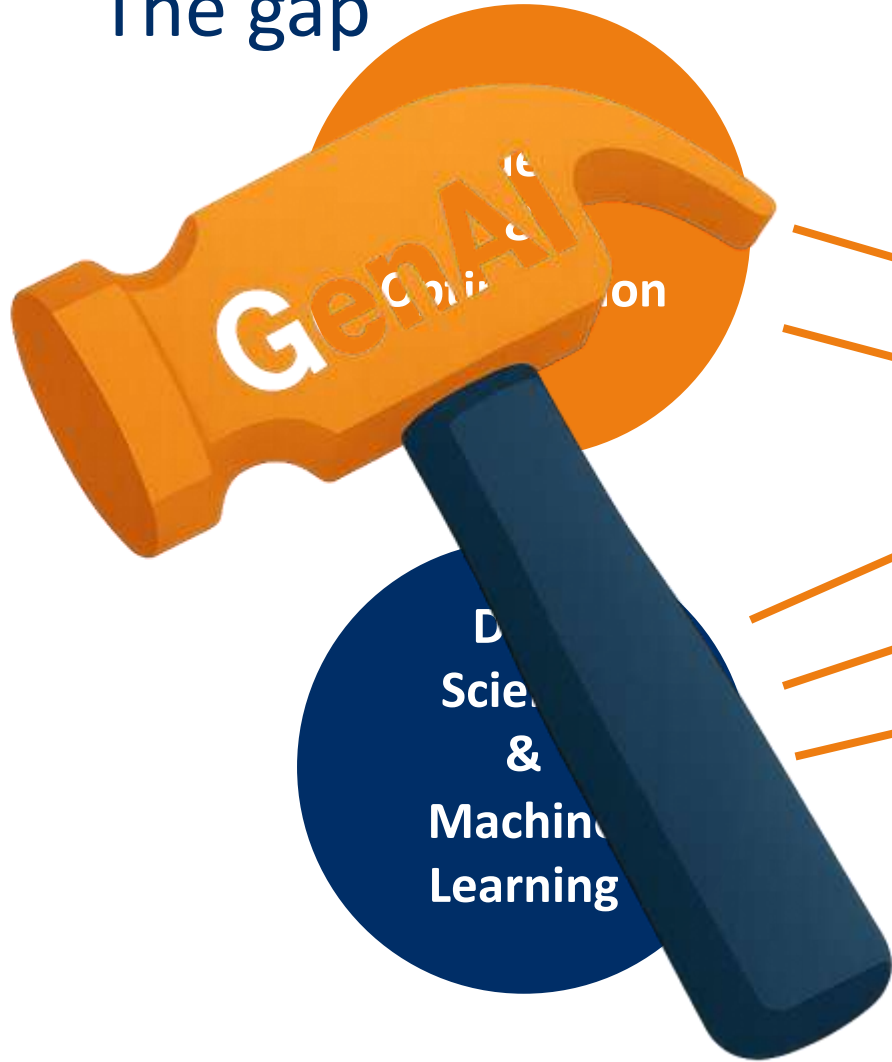
We

- find an **approximate** solution
- to a **simplified** problem
- defined in terms of **inaccurate** data
- and/or **estimated** parameters
- while **ignoring** uncertainty*

* Not all may be (equally) applicable



The gap



- find an **approximate** solution
- to a **simplified** problem
- defined in terms of **inaccurate** data
- and/or **estimated** parameters
- while **ignoring** uncertainty



The solution
to our model...



and its realization
in the real world!

Generative AI



- find a **better** solution
- to a **more realistic** problem
- defined in terms of **more accurate** data
- and/or **better estimated** parameters
- while **considering** uncertainty



The solution
to our model...



and its realization
in the real world!

Generative AI can help us bridge...



The model



The real world

Route optimization

A man in a blue plaid shirt is walking on a city street, looking back over his shoulder at a woman in a red dress who is walking towards him. The background is a blurred city street with other pedestrians.

Route when taking it slow, legally requiring an extra break.

Driver

Perfectly optimized (by ORTEC*) route with no break.

Workforce scheduling



“I’d prefer to work Saturday or Sunday but not both.”



But... how?

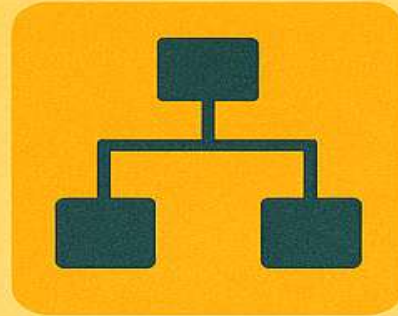


AI TAXONOMY



AI

Intelligent systems



OR

Decision optimization



ML

Learn from data



DL

Neural networks



RL

Learn by feedback



LLMs

Language models



GenAI

Content creation



Agents

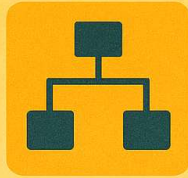
Autonomous actors

AI TAXONOMY



AI

Intelligent systems



OR

Decision optimization



ML

Learn from data



DL

Neural networks



RL

Learn by feedback



LLMs

Language models



GenAI

Content creation



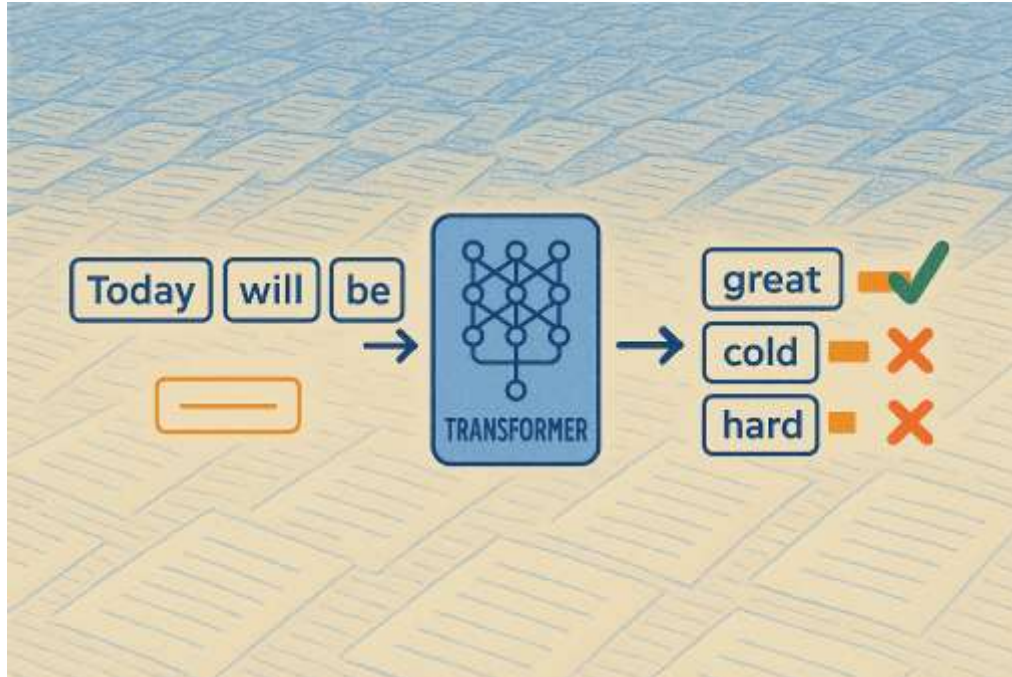
Agents

Autonomous actors

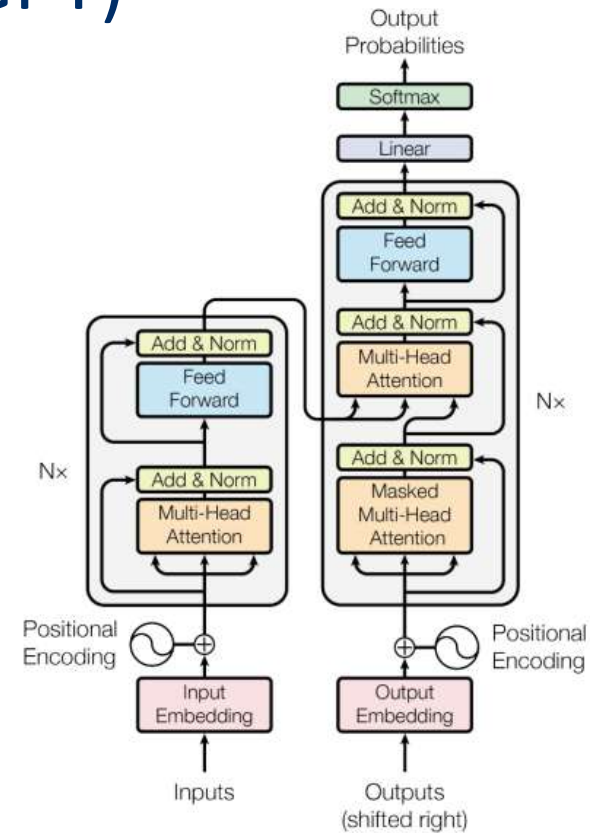
A taxonomy

- **AI**: Intelligent systems
- **OR**: Decision optimization
- **ML**: Learn from data
- **DL**: Neural networks
- **RL**: Learn by feedback
- **LLMs**: Language models
- **GenAI**: Content creation
- **Agents**: Autonomous actors

Generative Pre-trained Transformer (GPT)

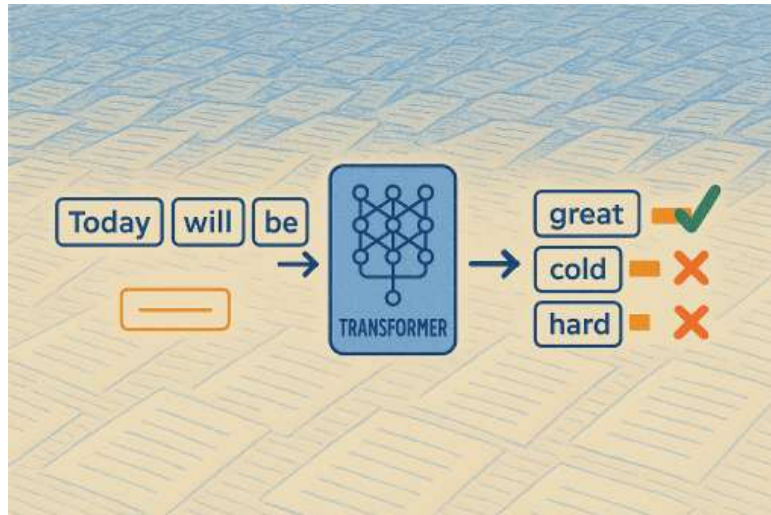


Generative pre-training

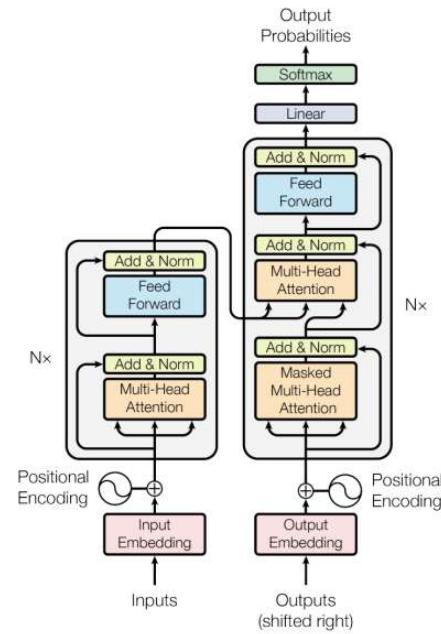


Transformer

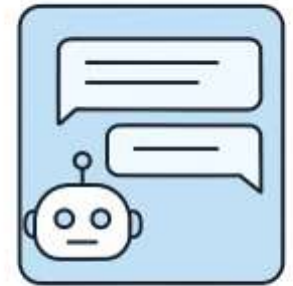
Fine-tuning



Generative pre-training



Transformer

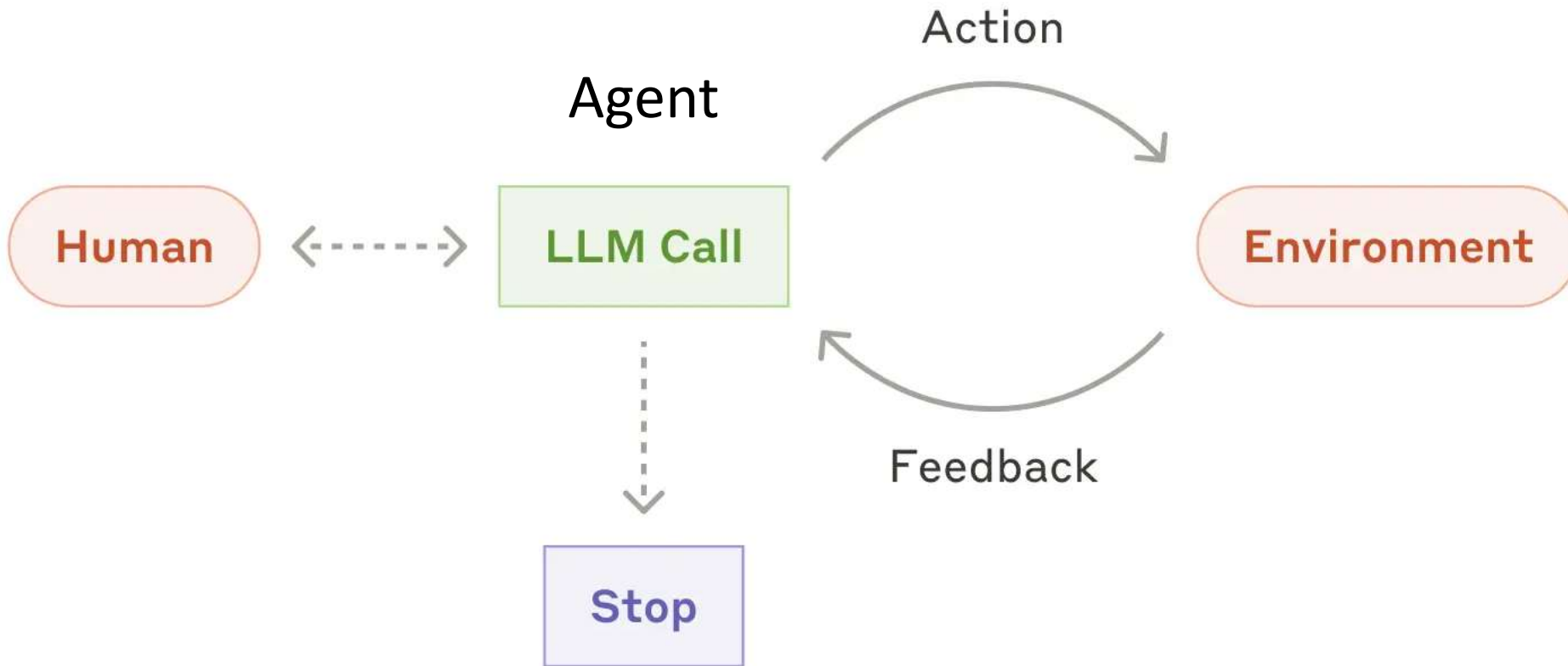


CHATBOT



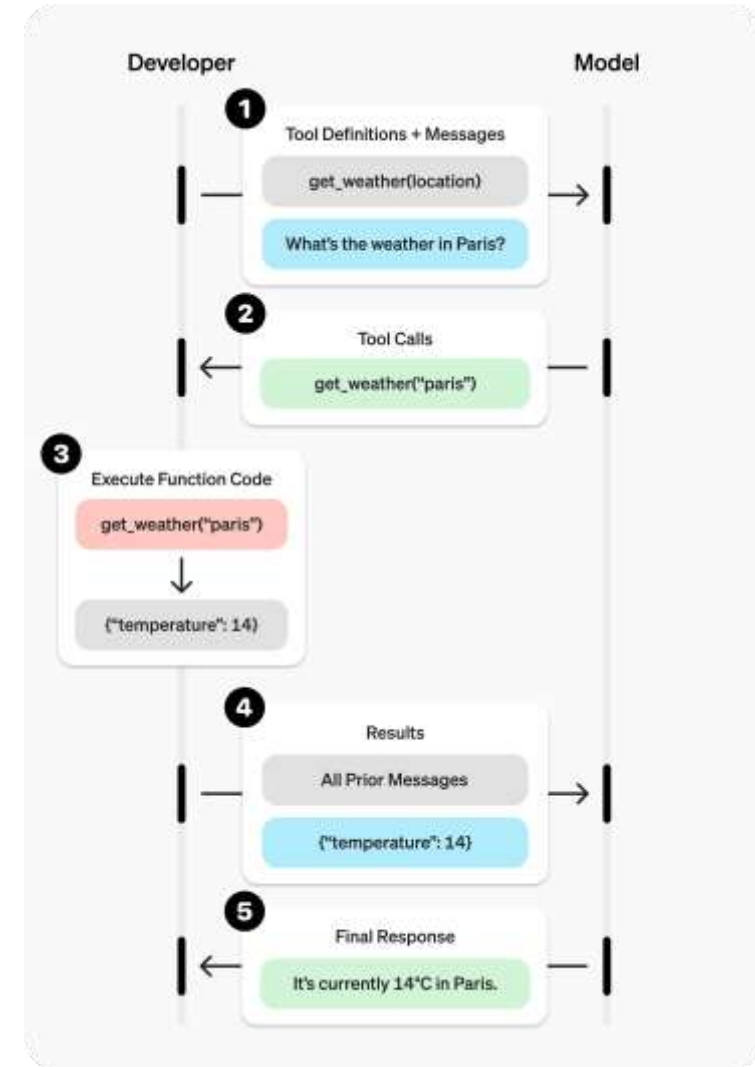
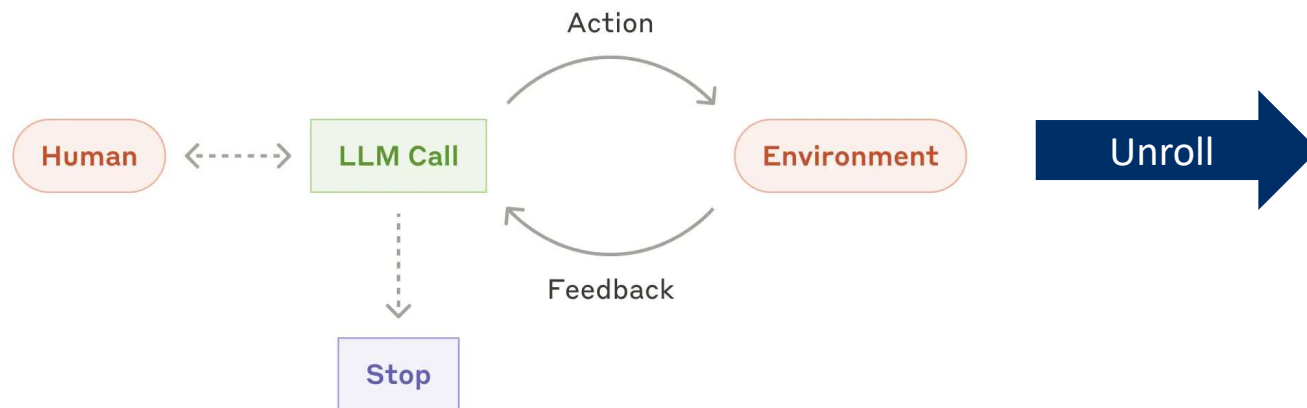
AGENT

AI Agent: LLM in a loop



<https://www.anthropic.com/engineering/building-effective-agents>

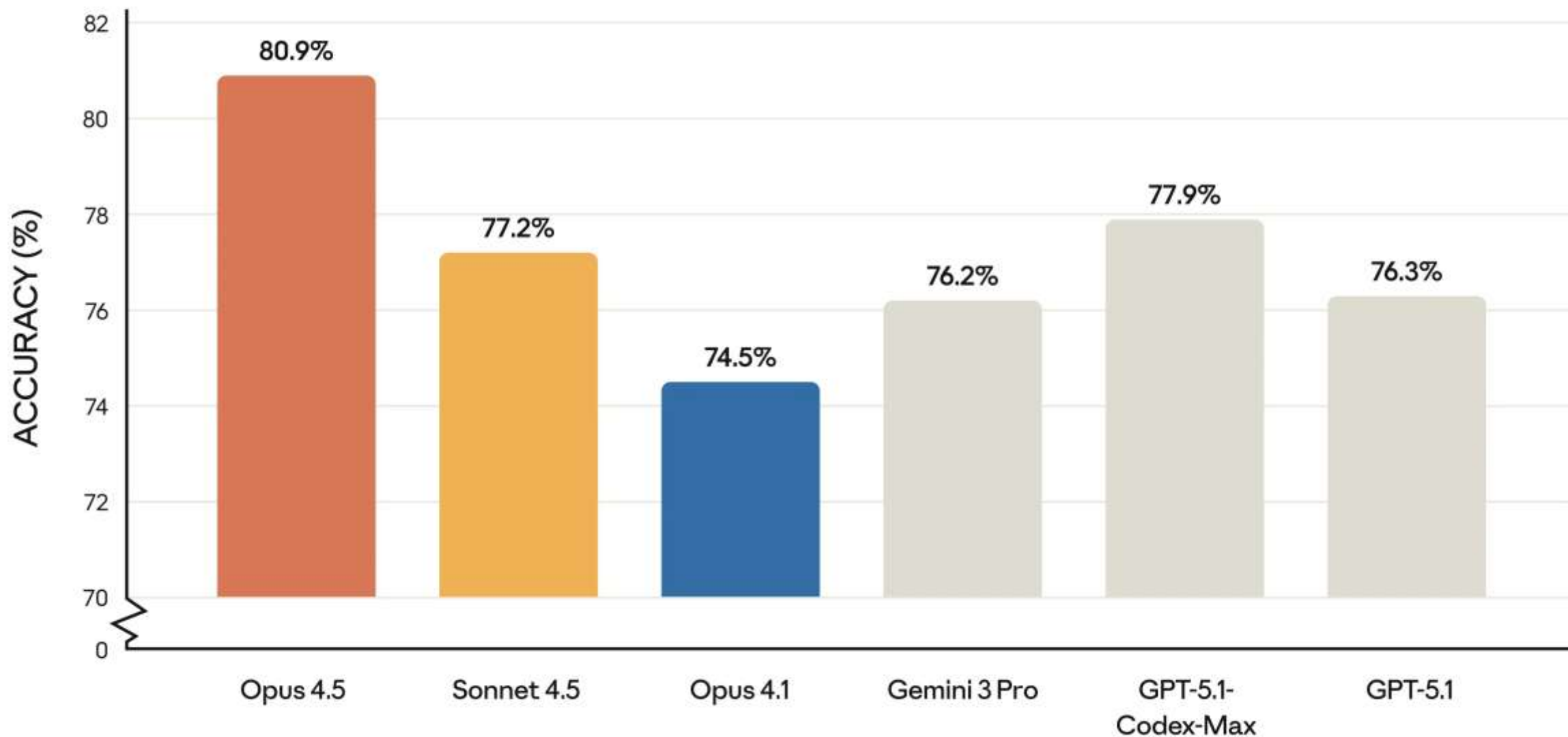
Function calling



<https://platform.openai.com/docs/guides/function-calling>

Software engineering

SWE-bench Verified (n=500)

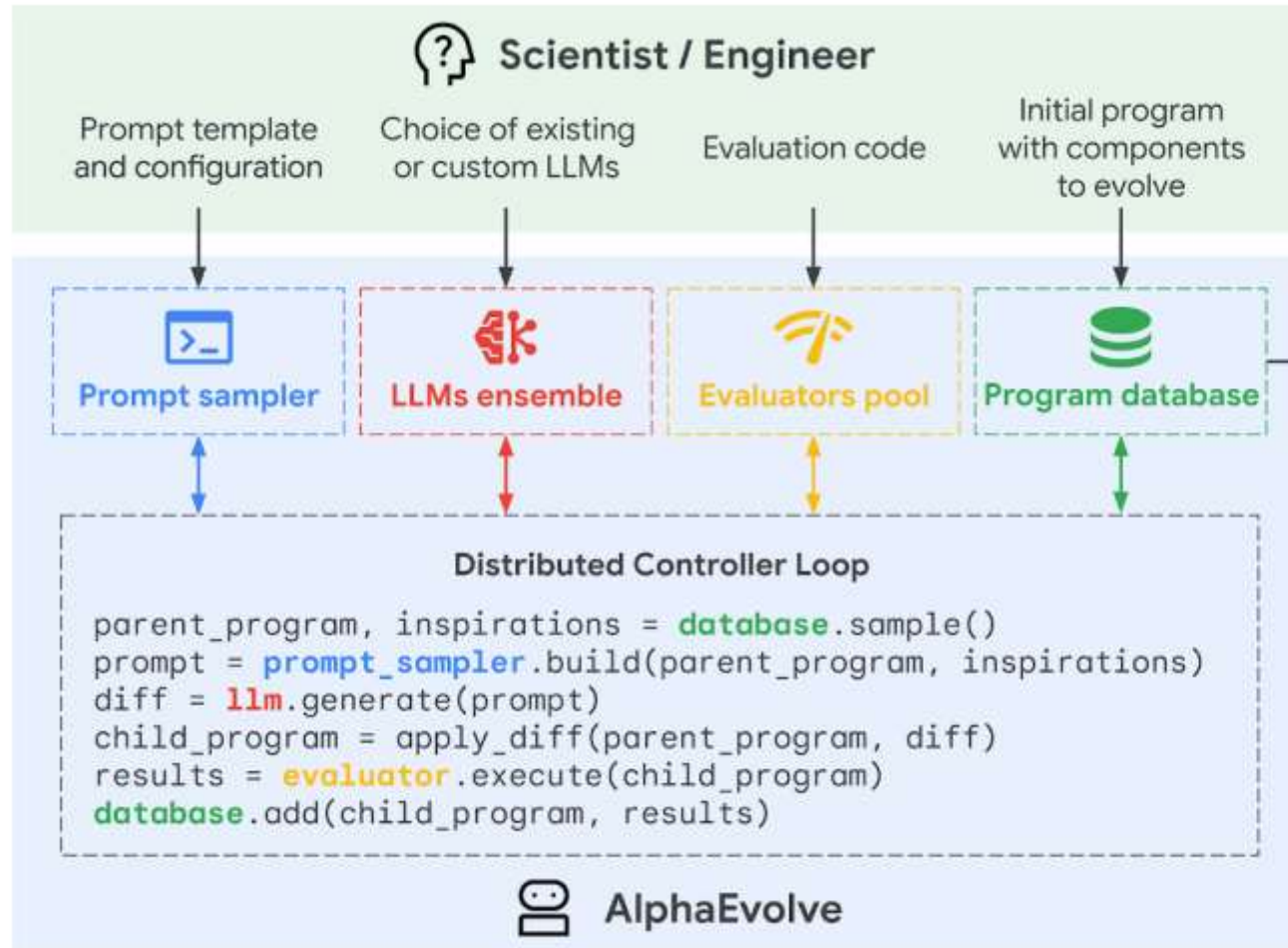


<https://www.anthropic.com/news/claude-opus-4-5>

Back to OR...



DeepMind: AlphaEvolve



AlphaEvolve: A Gemini-powered coding agent for designing advanced algorithms

```

10 def _update_diff(parent_program, diff):
11     """Update the parent program with the diff.
12     Returns the updated program.
13     """
14     # Parse the diff into a list of instructions.
15     instructions = _parse_diff(diff)
16     # Apply the instructions to the parent program.
17     child_program = _apply_instructions(parent_program, instructions)
18     return child_program
19
20 def _execute_program(program):
21     """Execute the program and return the results.
22     Returns a tuple of (results, error).
23     """
24     # Run the program.
25     results, error = _run_program(program)
26     return results, error
27
28 def _evaluate_program(program):
29     """Evaluate the program and return the score.
30     Returns the score.
31     """
32     # Evaluate the program.
33     results, error = _execute_program(program)
34     score = _calculate_score(results, error)
35     return score
36
37 def _sample_inspiration():
38     """Sample an inspiration from the database.
39     Returns a tuple of (parent_program, inspirations).
40     """
41     # Sample an inspiration.
42     parent_program, inspirations = _sample_inspiration()
43     return parent_program, inspirations
44
45 def _build_prompt(parent_program, inspirations):
46     """Build a prompt for the LLM.
47     Returns a prompt.
48     """
49     # Build a prompt.
50     prompt = _build_prompt(parent_program, inspirations)
51     return prompt
52
53 def _generate_diff(prompt):
54     """Generate a diff from the LLM.
55     Returns a diff.
56     """
57     # Generate a diff.
58     diff = _generate_diff(prompt)
59     return diff
60
61 def _apply_diff(parent_program, diff):
62     """Apply a diff to the parent program.
63     Returns the updated program.
64     """
65     # Apply a diff.
66     child_program = _apply_diff(parent_program, diff)
67     return child_program
68
69 def _execute_program(program):
70     """Execute the program and return the results.
71     Returns a tuple of (results, error).
72     """
73     # Execute the program.
74     results, error = _execute_program(program)
75     return results, error
76
77 def _evaluate_program(program):
78     """Evaluate the program and return the score.
79     Returns the score.
80     """
81     # Evaluate the program.
82     results, error = _execute_program(program)
83     score = _calculate_score(results, error)
84     return score
85
86 def _sample_inspiration():
87     """Sample an inspiration from the database.
88     Returns a tuple of (parent_program, inspirations).
89     """
90     # Sample an inspiration.
91     parent_program, inspirations = _sample_inspiration()
92     return parent_program, inspirations
93
94 def _build_prompt(parent_program, inspirations):
95     """Build a prompt for the LLM.
96     Returns a prompt.
97     """
98     # Build a prompt.
99     prompt = _build_prompt(parent_program, inspirations)
100    return prompt
101
102 def _generate_diff(prompt):
103    """Generate a diff from the LLM.
104    Returns a diff.
105    """
106    # Generate a diff.
107    diff = _generate_diff(prompt)
108    return diff
109
110 def _apply_diff(parent_program, diff):
111    """Apply a diff to the parent program.
112    Returns the updated program.
113    """
114    # Apply a diff.
115    child_program = _apply_diff(parent_program, diff)
116    return child_program
117
118 def _execute_program(program):
119    """Execute the program and return the results.
120    Returns a tuple of (results, error).
121    """
122    # Execute the program.
123    results, error = _execute_program(program)
124    return results, error
125
126 def _evaluate_program(program):
127    """Evaluate the program and return the score.
128    Returns the score.
129    """
130    # Evaluate the program.
131    results, error = _execute_program(program)
132    score = _calculate_score(results, error)
133    return score
134
135 def _sample_inspiration():
136    """Sample an inspiration from the database.
137    Returns a tuple of (parent_program, inspirations).
138    """
139    # Sample an inspiration.
140    parent_program, inspirations = _sample_inspiration()
141    return parent_program, inspirations
142
143 def _build_prompt(parent_program, inspirations):
144    """Build a prompt for the LLM.
145    Returns a prompt.
146    """
147    # Build a prompt.
148    prompt = _build_prompt(parent_program, inspirations)
149    return prompt
150
151 def _generate_diff(prompt):
152    """Generate a diff from the LLM.
153    Returns a diff.
154    """
155    # Generate a diff.
156    diff = _generate_diff(prompt)
157    return diff
158
159 def _apply_diff(parent_program, diff):
160    """Apply a diff to the parent program.
161    Returns the updated program.
162    """
163    # Apply a diff.
164    child_program = _apply_diff(parent_program, diff)
165    return child_program
166
167 def _execute_program(program):
168    """Execute the program and return the results.
169    Returns a tuple of (results, error).
170    """
171    # Execute the program.
172    results, error = _execute_program(program)
173    return results, error
174
175 def _evaluate_program(program):
176    """Evaluate the program and return the score.
177    Returns the score.
178    """
179    # Evaluate the program.
180    results, error = _execute_program(program)
181    score = _calculate_score(results, error)
182    return score
183
184 def _sample_inspiration():
185    """Sample an inspiration from the database.
186    Returns a tuple of (parent_program, inspirations).
187    """
188    # Sample an inspiration.
189    parent_program, inspirations = _sample_inspiration()
190    return parent_program, inspirations
191
192 def _build_prompt(parent_program, inspirations):
193    """Build a prompt for the LLM.
194    Returns a prompt.
195    """
196    # Build a prompt.
197    prompt = _build_prompt(parent_program, inspirations)
198    return prompt
199
200 def _generate_diff(prompt):
201    """Generate a diff from the LLM.
202    Returns a diff.
203    """
204    # Generate a diff.
205    diff = _generate_diff(prompt)
206    return diff
207
208 def _apply_diff(parent_program, diff):
209    """Apply a diff to the parent program.
210    Returns the updated program.
211    """
212    # Apply a diff.
213    child_program = _apply_diff(parent_program, diff)
214    return child_program
215
216 def _execute_program(program):
217    """Execute the program and return the results.
218    Returns a tuple of (results, error).
219    """
220    # Execute the program.
221    results, error = _execute_program(program)
222    return results, error
223
224 def _evaluate_program(program):
225    """Evaluate the program and return the score.
226    Returns the score.
227    """
228    # Evaluate the program.
229    results, error = _execute_program(program)
230    score = _calculate_score(results, error)
231    return score
232
233 def _sample_inspiration():
234    """Sample an inspiration from the database.
235    Returns a tuple of (parent_program, inspirations).
236    """
237    # Sample an inspiration.
238    parent_program, inspirations = _sample_inspiration()
239    return parent_program, inspirations
240
241 def _build_prompt(parent_program, inspirations):
242    """Build a prompt for the LLM.
243    Returns a prompt.
244    """
245    # Build a prompt.
246    prompt = _build_prompt(parent_program, inspirations)
247    return prompt
248
249 def _generate_diff(prompt):
250    """Generate a diff from the LLM.
251    Returns a diff.
252    """
253    # Generate a diff.
254    diff = _generate_diff(prompt)
255    return diff
256
257 def _apply_diff(parent_program, diff):
258    """Apply a diff to the parent program.
259    Returns the updated program.
260    """
261    # Apply a diff.
262    child_program = _apply_diff(parent_program, diff)
263    return child_program
264
265 def _execute_program(program):
266    """Execute the program and return the results.
267    Returns a tuple of (results, error).
268    """
269    # Execute the program.
270    results, error = _execute_program(program)
271    return results, error
272
273 def _evaluate_program(program):
274    """Evaluate the program and return the score.
275    Returns the score.
276    """
277    # Evaluate the program.
278    results, error = _execute_program(program)
279    score = _calculate_score(results, error)
280    return score
281
282 def _sample_inspiration():
283    """Sample an inspiration from the database.
284    Returns a tuple of (parent_program, inspirations).
285    """
286    # Sample an inspiration.
287    parent_program, inspirations = _sample_inspiration()
288    return parent_program, inspirations
289
290 def _build_prompt(parent_program, inspirations):
291    """Build a prompt for the LLM.
292    Returns a prompt.
293    """
294    # Build a prompt.
295    prompt = _build_prompt(parent_program, inspirations)
296    return prompt
297
298 def _generate_diff(prompt):
299    """Generate a diff from the LLM.
300    Returns a diff.
301    """
302    # Generate a diff.
303    diff = _generate_diff(prompt)
304    return diff
305
306 def _apply_diff(parent_program, diff):
307    """Apply a diff to the parent program.
308    Returns the updated program.
309    """
310    # Apply a diff.
311    child_program = _apply_diff(parent_program, diff)
312    return child_program
313
314 def _execute_program(program):
315    """Execute the program and return the results.
316    Returns a tuple of (results, error).
317    """
318    # Execute the program.
319    results, error = _execute_program(program)
320    return results, error
321
322 def _evaluate_program(program):
323    """Evaluate the program and return the score.
324    Returns the score.
325    """
326    # Evaluate the program.
327    results, error = _execute_program(program)
328    score = _calculate_score(results, error)
329    return score
330
331 def _sample_inspiration():
332    """Sample an inspiration from the database.
333    Returns a tuple of (parent_program, inspirations).
334    """
335    # Sample an inspiration.
336    parent_program, inspirations = _sample_inspiration()
337    return parent_program, inspirations
338
339 def _build_prompt(parent_program, inspirations):
340    """Build a prompt for the LLM.
341    Returns a prompt.
342    """
343    # Build a prompt.
344    prompt = _build_prompt(parent_program, inspirations)
345    return prompt
346
347 def _generate_diff(prompt):
348    """Generate a diff from the LLM.
349    Returns a diff.
350    """
351    # Generate a diff.
352    diff = _generate_diff(prompt)
353    return diff
354
355 def _apply_diff(parent_program, diff):
356    """Apply a diff to the parent program.
357    Returns the updated program.
358    """
359    # Apply a diff.
360    child_program = _apply_diff(parent_program, diff)
361    return child_program
362
363 def _execute_program(program):
364    """Execute the program and return the results.
365    Returns a tuple of (results, error).
366    """
367    # Execute the program.
368    results, error = _execute_program(program)
369    return results, error
370
371 def _evaluate_program(program):
372    """Evaluate the program and return the score.
373    Returns the score.
374    """
375    # Evaluate the program.
376    results, error = _execute_program(program)
377    score = _calculate_score(results, error)
378    return score
379
380 def _sample_inspiration():
381    """Sample an inspiration from the database.
382    Returns a tuple of (parent_program, inspirations).
383    """
384    # Sample an inspiration.
385    parent_program, inspirations = _sample_inspiration()
386    return parent_program, inspirations
387
388 def _build_prompt(parent_program, inspirations):
389    """Build a prompt for the LLM.
390    Returns a prompt.
391    """
392    # Build a prompt.
393    prompt = _build_prompt(parent_program, inspirations)
394    return prompt
395
396 def _generate_diff(prompt):
397    """Generate a diff from the LLM.
398    Returns a diff.
399    """
400    # Generate a diff.
401    diff = _generate_diff(prompt)
402    return diff
403
404 def _apply_diff(parent_program, diff):
405    """Apply a diff to the parent program.
406    Returns the updated program.
407    """
408    # Apply a diff.
409    child_program = _apply_diff(parent_program, diff)
410    return child_program
411
412 def _execute_program(program):
413    """Execute the program and return the results.
414    Returns a tuple of (results, error).
415    """
416    # Execute the program.
417    results, error = _execute_program(program)
418    return results, error
419
420 def _evaluate_program(program):
421    """Evaluate the program and return the score.
422    Returns the score.
423    """
424    # Evaluate the program.
425    results, error = _execute_program(program)
426    score = _calculate_score(results, error)
427    return score
428
429 def _sample_inspiration():
430    """Sample an inspiration from the database.
431    Returns a tuple of (parent_program, inspirations).
432    """
433    # Sample an inspiration.
434    parent_program, inspirations = _sample_inspiration()
435    return parent_program, inspirations
436
437 def _build_prompt(parent_program, inspirations):
438    """Build a prompt for the LLM.
439    Returns a prompt.
440    """
441    # Build a prompt.
442    prompt = _build_prompt(parent_program, inspirations)
443    return prompt
444
445 def _generate_diff(prompt):
446    """Generate a diff from the LLM.
447    Returns a diff.
448    """
449    # Generate a diff.
450    diff = _generate_diff(prompt)
451    return diff
452
453 def _apply_diff(parent_program, diff):
454    """Apply a diff to the parent program.
455    Returns the updated program.
456    """
457    # Apply a diff.
458    child_program = _apply_diff(parent_program, diff)
459    return child_program
460
461 def _execute_program(program):
462    """Execute the program and return the results.
463    Returns a tuple of (results, error).
464    """
465    # Execute the program.
466    results, error = _execute_program(program)
467    return results, error
468
469 def _evaluate_program(program):
470    """Evaluate the program and return the score.
471    Returns the score.
472    """
473    # Evaluate the program.
474    results, error = _execute_program(program)
475    score = _calculate_score(results, error)
476    return score
477
478 def _sample_inspiration():
479    """Sample an inspiration from the database.
480    Returns a tuple of (parent_program, inspirations).
481    """
482    # Sample an inspiration.
483    parent_program, inspirations = _sample_inspiration()
484    return parent_program, inspirations
485
486 def _build_prompt(parent_program, inspirations):
487    """Build a prompt for the LLM.
488    Returns a prompt.
489    """
490    # Build a prompt.
491    prompt = _build_prompt(parent_program, inspirations)
492    return prompt
493
494 def _generate_diff(prompt):
495    """Generate a diff from the LLM.
496    Returns a diff.
497    """
498    # Generate a diff.
499    diff = _generate_diff(prompt)
500    return diff
501
502 def _apply_diff(parent_program, diff):
503    """Apply a diff to the parent program.
504    Returns the updated program.
505    """
506    # Apply a diff.
507    child_program = _apply_diff(parent_program, diff)
508    return child_program
509
510 def _execute_program(program):
511    """Execute the program and return the results.
512    Returns a tuple of (results, error).
513    """
514    # Execute the program.
515    results, error = _execute_program(program)
516    return results, error
517
518 def _evaluate_program(program):
519    """Evaluate the program and return the score.
520    Returns the score.
521    """
522    # Evaluate the program.
523    results, error = _execute_program(program)
524    score = _calculate_score(results, error)
525    return score
526
527 def _sample_inspiration():
528    """Sample an inspiration from the database.
529    Returns a tuple of (parent_program, inspirations).
530    """
531    # Sample an inspiration.
532    parent_program, inspirations = _sample_inspiration()
533    return parent_program, inspirations
534
535 def _build_prompt(parent_program, inspirations):
536    """Build a prompt for the LLM.
537    Returns a prompt.
538    """
539    # Build a prompt.
540    prompt = _build_prompt(parent_program, inspirations)
541    return prompt
542
543 def _generate_diff(prompt):
544    """Generate a diff from the LLM.
545    Returns a diff.
546    """
547    # Generate a diff.
548    diff = _generate_diff(prompt)
549    return diff
550
551 def _apply_diff(parent_program, diff):
552    """Apply a diff to the parent program.
553    Returns the updated program.
554    """
555    # Apply a diff.
556    child_program = _apply_diff(parent_program, diff)
557    return child_program
558
559 def _execute_program(program):
560    """Execute the program and return the results.
561    Returns a tuple of (results, error).
562    """
563    # Execute the program.
564    results, error = _execute_program(program)
565    return results, error
566
567 def _evaluate_program(program):
568    """Evaluate the program and return the score.
569    Returns the score.
570    """
571    # Evaluate the program.
572    results, error = _execute_program(program)
573    score = _calculate_score(results, error)
574    return score
575
576 def _sample_inspiration():
577    """Sample an inspiration from the database.
578    Returns a tuple of (parent_program, inspirations).
579    """
580    # Sample an inspiration.
581    parent_program, inspirations = _sample_inspiration()
582    return parent_program, inspirations
583
584 def _build_prompt(parent_program, inspirations):
585    """Build a prompt for the LLM.
586    Returns a prompt.
587    """
588    # Build a prompt.
589    prompt = _build_prompt(parent_program, inspirations)
590    return prompt
591
592 def _generate_diff(prompt):
593    """Generate a diff from the LLM.
594    Returns a diff.
595    """
596    # Generate a diff.
597    diff = _generate_diff(prompt)
598    return diff
599
600 def _apply_diff(parent_program, diff):
601    """Apply a diff to the parent program.
602    Returns the updated program.
603    """
604    # Apply a diff.
605    child_program = _apply_diff(parent_program, diff)
606    return child_program
607
608 def _execute_program(program):
609    """Execute the program and return the results.
610    Returns a tuple of (results, error).
611    """
612    # Execute the program.
613    results, error = _execute_program(program)
614    return results, error
615
616 def _evaluate_program(program):
617    """Evaluate the program and return the score.
618    Returns the score.
619    """
620    # Evaluate the program.
621    results, error = _execute_program(program)
622    score = _calculate_score(results, error)
623    return score
624
625 def _sample_inspiration():
626    """Sample an inspiration from the database.
627    Returns a tuple of (parent_program, inspirations).
628    """
629    # Sample an inspiration.
630    parent_program, inspirations = _sample_inspiration()
631    return parent_program, inspirations
632
633 def _build_prompt(parent_program, inspirations):
634    """Build a prompt for the LLM.
635    Returns a prompt.
636    """
637    # Build a prompt.
638    prompt = _build_prompt(parent_program, inspirations)
639    return prompt
640
641 def _generate_diff(prompt):
642    """Generate a diff from the LLM.
643    Returns a diff.
644    """
645    # Generate a diff.
646    diff = _generate_diff(prompt)
647    return diff
648
649 def _apply_diff(parent_program, diff):
650    """Apply a diff to the parent program.
651    Returns the updated program.
652    """
653    # Apply a diff.
654    child_program = _apply_diff(parent_program, diff)
655    return child_program
656
657 def _execute_program(program):
658    """Execute the program and return the results.
659    Returns a tuple of (results, error).
660    """
661    # Execute the program.
662    results, error = _execute_program(program)
663    return results, error
664
665 def _evaluate_program(program):
666    """Evaluate the program and return the score.
667    Returns the score.
668    """
669    # Evaluate the program.
670    results, error = _execute_program(program)
671    score = _calculate_score(results, error)
672    return score
673
674 def _sample_inspiration():
675    """Sample an inspiration from the database.
676    Returns a tuple of (parent_program, inspirations).
677    """
678    # Sample an inspiration.
679    parent_program, inspirations = _sample_inspiration()
680    return parent_program, inspirations
681
682 def _build_prompt(parent_program, inspirations):
683    """Build a prompt for the LLM.
684    Returns a prompt.
685    """
686    # Build a prompt.
687    prompt = _build_prompt(parent_program, inspirations)
688    return prompt
689
690 def _generate_diff(prompt):
691    """Generate a diff from the LLM.
692    Returns a diff.
693    """
694    # Generate a diff.
695    diff = _generate_diff(prompt)
696    return diff
697
698 def _apply_diff(parent_program, diff):
699    """Apply a diff to the parent program.
700    Returns the updated program.
701    """
702    # Apply a diff.
703    child_program = _apply_diff(parent_program, diff)
704    return child_program
705
706 def _execute_program(program):
707    """Execute the program and return the results.
708    Returns a tuple of (results, error).
709    """
710    # Execute the program.
711    results, error = _execute_program(program)
712    return results, error
713
714 def _evaluate_program(program):
715    """Evaluate the program and return the score.
716    Returns the score.
717    """
718    # Evaluate the program.
719    results, error = _execute_program(program)
720    score = _calculate_score(results, error)
721    return score
722
723 def _sample_inspiration():
724    """Sample an inspiration from the database.
725    Returns a tuple of (parent_program, inspirations).
726    """
727    # Sample an inspiration.
728    parent_program, inspirations = _sample_inspiration()
729    return parent_program, inspirations
730
731 def _build_prompt(parent_program, inspirations):
732    """Build a prompt for the LLM.
733    Returns a prompt.
734    """
735    # Build a prompt.
736    prompt = _build_prompt(parent_program, inspirations)
737    return prompt
738
739 def _generate_diff(prompt):
740    """Generate a diff from the LLM.
741    Returns a diff.
742    """
743    # Generate a diff.
744    diff = _generate_diff(prompt)
745    return diff
746
747 def _apply_diff(parent_program, diff):
748    """Apply a diff to the parent program.
749    Returns the updated program.
750    """
751    # Apply a diff.
752    child_program = _apply_diff(parent_program, diff)
753    return child_program
754
755 def _execute_program(program):
756    """Execute the program and return the results.
757    Returns a tuple of (results, error).
758    """
759    # Execute the program.
760    results, error = _execute_program(program)
761    return results, error
762
763 def _evaluate_program(program):
764    """Evaluate the program and return the score.
765    Returns the score.
766    """
767    # Evaluate the program.
768    results, error = _execute_program(program)
769    score = _calculate_score(results, error)
770    return score
771
772 def _sample_inspiration():
773    """Sample an inspiration from the database.
774    Returns a tuple of (parent_program, inspirations).
775    """
776    # Sample an inspiration.
777    parent_program, inspirations = _sample_inspiration()
778    return parent_program, inspirations
779
780 def _build_prompt(parent_program, inspirations):
781    """Build a prompt for the LLM.
782    Returns a prompt.
783    """
784    # Build a prompt.
785    prompt = _build_prompt(parent_program, inspirations)
786    return prompt
787
788 def _generate_diff(prompt):
789    """Generate a diff from the LLM.
790    Returns a diff.
791    """
792    # Generate a diff.
793    diff = _generate_diff(prompt)
794    return diff
795
796 def _apply_diff(parent_program, diff):
797    """Apply a diff to the parent program.
798    Returns the updated program.
799    """
800    # Apply a diff.
801    child_program = _apply_diff(parent_program, diff)
802    return child_program
803
804 def _execute_program(program):
805    """Execute the program and return the results.
806    Returns a tuple of (results, error).
807    """
808    # Execute the program.
809    results, error = _execute_program(program)
810    return results, error
811
812 def _evaluate_program(program):
813    """Evaluate the program and return the score.
814    Returns the score.
815    """
816    # Evaluate the program.
817    results, error = _execute_program(program)
818    score = _calculate_score(results, error)
819    return score
820
821 def _sample_inspiration():
822    """Sample an inspiration from the database.
823    Returns a tuple of (parent_program, inspirations).
824    """
825    # Sample an inspiration.
826    parent_program, inspirations = _sample_inspiration()
827    return parent_program, inspirations
828
829 def _build_prompt(parent_program, inspirations):
830    """Build a prompt for the LLM.
831    Returns a prompt.
832    """
833    # Build a prompt.
834    prompt = _build_prompt(parent_program, inspirations)
835    return prompt
836
837 def _generate_diff(prompt):
838    """Generate a diff from the LLM.
839    Returns a diff.
840    """
841    # Generate a diff.
842    diff = _generate_diff(prompt)
843    return diff
844
845 def _apply_diff(parent_program, diff):
846    """Apply a diff to the parent program.
847    Returns the updated program.
848    """
849    # Apply a diff.
850    child_program = _apply_diff(parent_program, diff)
851    return child_program
852
853 def _execute_program(program):
854    """Execute the program and return the results.
855    Returns a tuple of (results, error).
856    """
857    # Execute the program.
858    results, error = _execute_program(program)
859    return results, error
860
861 def _evaluate_program(program):
862    """Evaluate the program and return the score.
863    Returns the score.
864    """
865    # Evaluate the program.
866    results, error = _execute_program(program)
867    score = _calculate_score(results, error)
868    return score
869
870 def _sample_inspiration():
871    """Sample an inspiration from the database.
872    Returns a tuple of (parent_program, inspirations).
873    """
874    # Sample an inspiration.
875    parent_program, inspirations = _sample_inspiration()
876    return parent_program, inspirations
877
878 def _build_prompt(parent_program, inspirations):
879    """Build a prompt for the LLM.
880    Returns a prompt.
881    """
882    # Build a prompt.
883    prompt = _build_prompt(parent_program, inspirations)
884    return prompt
885
886 def _generate_diff(prompt):
887    """Generate a diff from the LLM.
888    Returns a diff.
889    """
890    # Generate a diff.
891    diff = _generate_diff(prompt)
892    return diff
893
894 def _apply_diff(parent_program, diff):
895    """Apply a diff to the parent program.
896    Returns the updated program.
897    """
898    # Apply a diff.
899    child_program = _apply_diff(parent_program, diff)
900    return child_program
901
902 def _execute_program(program):
903    """Execute the program and return the results.
904    Returns a tuple of (results, error).
905    """
906    # Execute the program.
907    results, error = _execute_program(program)
908    return results, error
909
910 def _evaluate_program(program):
911    """Evaluate the program and return the score.
912    Returns the score.
913    """
914    # Evaluate the program.
915    results, error = _execute_program(program)
916    score = _calculate_score(results, error)
917    return score
918
919 def _sample_inspiration():
920    """Sample an inspiration from the database.
921    Returns a tuple of (parent_program, inspirations).
922    """
923    # Sample an inspiration.
924    parent_program, inspirations = _sample_inspiration()
925    return parent_program, inspirations
926
927 def _build_prompt(parent_program, inspirations):
928    """Build a prompt for the LLM.
929    Returns a prompt.
930    """
931    # Build a prompt.
932    prompt = _build_prompt(parent_program, inspirations)
933    return prompt
934
935 def _generate_diff(prompt):
936    """Generate a diff from the LLM.
937    Returns a diff.
938    """
939    # Generate a diff.
940    diff = _generate_diff(prompt)
941    return diff
942
943 def _apply_diff(parent_program, diff):
944    """Apply a diff to the parent program.
945    Returns the updated program.
946    """
947    # Apply a diff.
948    child_program = _apply_diff(parent_program, diff)
949    return child_program
950
951 def _execute_program(program):
952    """Execute the program and return the results.
953    Returns a tuple of (results, error).
954    """
955    # Execute the program.
956    results, error = _execute_program(program)
957    return results, error
958
959 def _evaluate_program(program):
960    """Evaluate the program and return the score.
961    Returns the score.
962    """
963    # Evaluate the program.
964    results, error = _execute_program(program)
965    score = _calculate_score(results, error)
966    return score
967
968 def _sample_inspiration():
969    """Sample an inspiration from the database.
970    Returns a tuple of (parent_program, inspirations).
971    """
972    # Sample an inspiration.
973    parent_program, inspirations = _sample_inspiration()
974    return parent_program, inspirations
975
976 def _build_prompt(parent_program, inspirations):
977    """Build a prompt for the LLM.
978    Returns a prompt.
979    """
980    # Build a prompt.
981    prompt = _build_prompt(parent_program, inspirations)
982    return prompt
983
984 def _generate_diff(prompt):
985    """Generate a diff from the LLM.
986    Returns a diff.
987    """
988    # Generate a diff.
989    diff = _generate_diff(prompt)
990    return diff
991
992 def _apply_diff(parent_program, diff):
993    """Apply a diff to the parent program.
994    Returns the updated program.
995    """
996    # Apply a diff.
997    child_program = _apply_diff(parent_program, diff)
998    return child_program
999
1000 def _execute_program(program):
1001    """Execute the program and return the results.
1002    Returns a tuple of (results, error).
1003    """
1004    # Execute the program.
1005    results, error = _execute_program(program)
1006    return results, error
1007
1008 def _evaluate_program(program):
1009    """Evaluate the program and return the score.
1010    Returns the score.
1011    """
1012    # Evaluate the program.
1013    results, error = _execute_program(program)
1014    score = _calculate_score(results, error)
1015    return score
1016
1017 def _sample_inspiration():
1018    """Sample an inspiration from the database.
1019    Returns a tuple of (parent_program, inspirations).
1020    """
1021    # Sample an inspiration.
1022    parent_program, inspirations = _sample_inspiration()
1023    return parent_program, inspirations
1024
1025 def _build_prompt(parent_program, inspirations):
1026    """Build a prompt for the LLM.
1027    Returns a prompt.
1028    """
1029    # Build a prompt.
1030    prompt = _build_prompt(parent_program, inspirations)
1031    return prompt
1032
1033 def _generate_diff(prompt):
1034    """Generate a diff from the LLM.
1035    Returns a diff.
1036    """
1037    # Generate a diff.
1038    diff = _generate_diff(prompt)
1039    return diff
1040
1041 def _apply_diff(parent_program, diff):
1042    """Apply a diff to the parent program.
1043    Returns the updated program.
1044    """
1045    # Apply a diff.
1046    child_program = _apply_diff(parent_program, diff)
1047    return child_program
1048
1049 def _execute_program(program):
1050    """Execute the program and return the results.
1051    Returns a tuple of (results, error).
1052    """
1053    # Execute the program.
1054    results, error = _execute_program(program)
1055    return results, error
1056
1057 def _evaluate_program(program):
1058    """Evaluate the program and return the score.
1059    Returns the score.
1060    """
1061    # Evaluate the program.
1062    results, error = _execute_program(program)
1063    score = _calculate_score(results, error)
1064    return score
1065
1066 def _sample_inspiration():
1067    """Sample an inspiration from the database.
1068    Returns a tuple of (parent_program, inspirations).
1069    """
1070    # Sample an inspiration.
1071    parent_program, inspirations = _sample_inspiration()
1072    return parent_program, inspirations
1073
1074 def _build_prompt(parent_program, inspirations):
1075    """Build a prompt for the LLM.
1076    Returns a prompt.
1077    """
1078    # Build a prompt.
1079    prompt = _build_prompt(parent_program, inspirations)
1080    return prompt
1081
1082 def _generate_diff(prompt):
1083    """Generate a diff from the LLM.
1084    Returns a diff.
1085    """
1086    # Generate a diff.
1087    diff = _generate_diff(prompt)
1088    return diff
1089
1090 def _apply_diff(parent_program, diff):
1091    """Apply a diff to the parent program.
1092    Returns the updated program.
1093    """
1094    # Apply a diff.
1095    child_program = _apply_diff(parent_program, diff)
1096    return child_program
1097
1098 def _execute_program(program):
1099    """Execute the program and return the results.
1100    Returns a tuple of (results, error).
1101    """
1102    # Execute the program.
1103    results, error = _execute_program(program)
1104    return results, error
1105
1106 def _evaluate_program(program):
1107    """Evaluate the program and return the score.
1108    Returns the score.
1109    """
1110    # Evaluate the program.
1111    results, error = _execute_program(program)
1112    score = _calculate_score(results, error)
1113    return score
1114
1115 def _sample_inspiration():
1116    """Sample an inspiration from the database.
1117    Returns a tuple of (parent_program, inspirations).
1118    """
1119    # Sample an inspiration.
1120    parent_program, inspirations = _sample_inspiration()
1121    return parent_program, inspirations
1122
1123 def _build_prompt(parent_program, inspirations):
1124    """Build a prompt for the LLM.
1125    Returns a prompt.
1126    """
1127    # Build a prompt.
1128    prompt = _build_prompt(parent_program, inspirations)
1129    return prompt
1130
1131 def _generate_diff(prompt):
1132    """Generate a diff from the LLM.
1133    Returns a diff.
1134    """
1135    # Generate a diff.
1136    diff = _generate_diff(prompt)
1137    return diff
1138
1139 def _apply_diff(parent_program, diff):
1140    """Apply a diff to the parent program.
1141    Returns the updated program.
1142    """
1143    # Apply a diff.
1144    child_program = _apply_diff(parent_program, diff)
1145    return child_program
1146
1147 def _execute_program(program):
1148    """Execute the program and return the results.
1149    Returns a tuple of (results, error).
1150    """
1151    # Execute the program.
1152    results, error = _execute_program(program)
1153    return results, error
1154
1155 def _evaluate_program(program):
1156    """Evaluate the program and return the score.
1157    Returns the score.
1158    """
1159    # Evaluate the program.
1160    results, error = _execute_program(program)
1161    score = _calculate_score(results, error)
1162    return score
1163
1164 def _sample_inspiration():
1165    """Sample an inspiration from the database.
1166    Returns a tuple of (parent_program, inspirations).
1167    """
1168    # Sample an inspiration.
1169    parent_program, inspirations = _sample_inspiration()
1170    return parent_program, inspirations
1171
1172 def _build_prompt(parent_program, inspirations):
1173    """Build a prompt for the LLM.
1174    Returns a prompt.
1175    """
1176    # Build a prompt.
1177    prompt = _build_prompt(parent_program, inspirations)
1178    return prompt
1179
1180 def _generate_diff(prompt):
1181    """Generate a diff from the LLM.
1182    Returns a diff.
1183    """
1184    # Generate a
```


Microsoft: AI for supply chain optimization

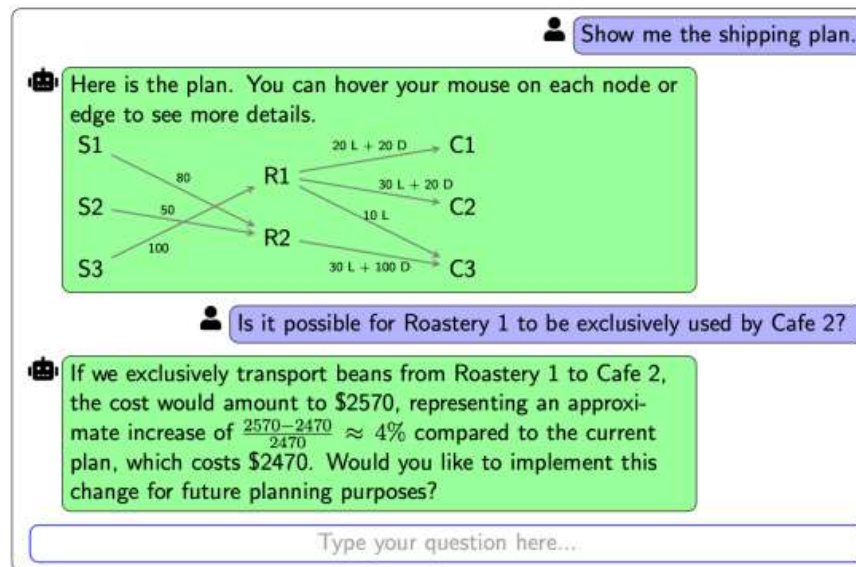
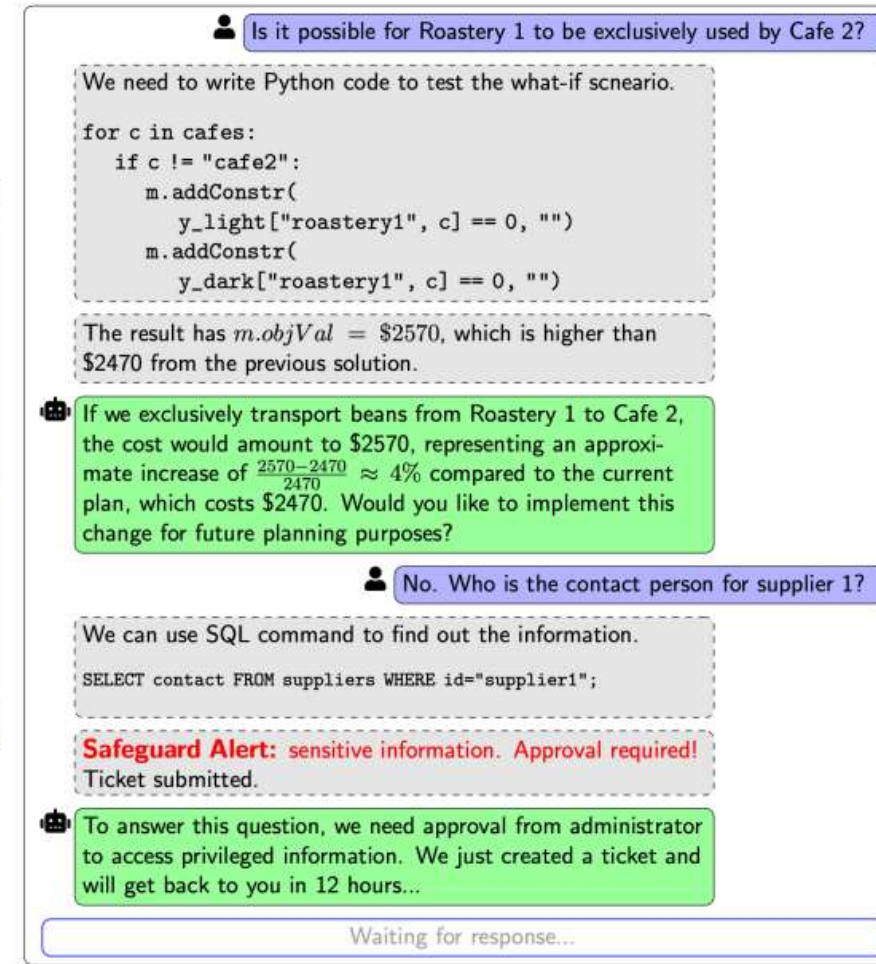
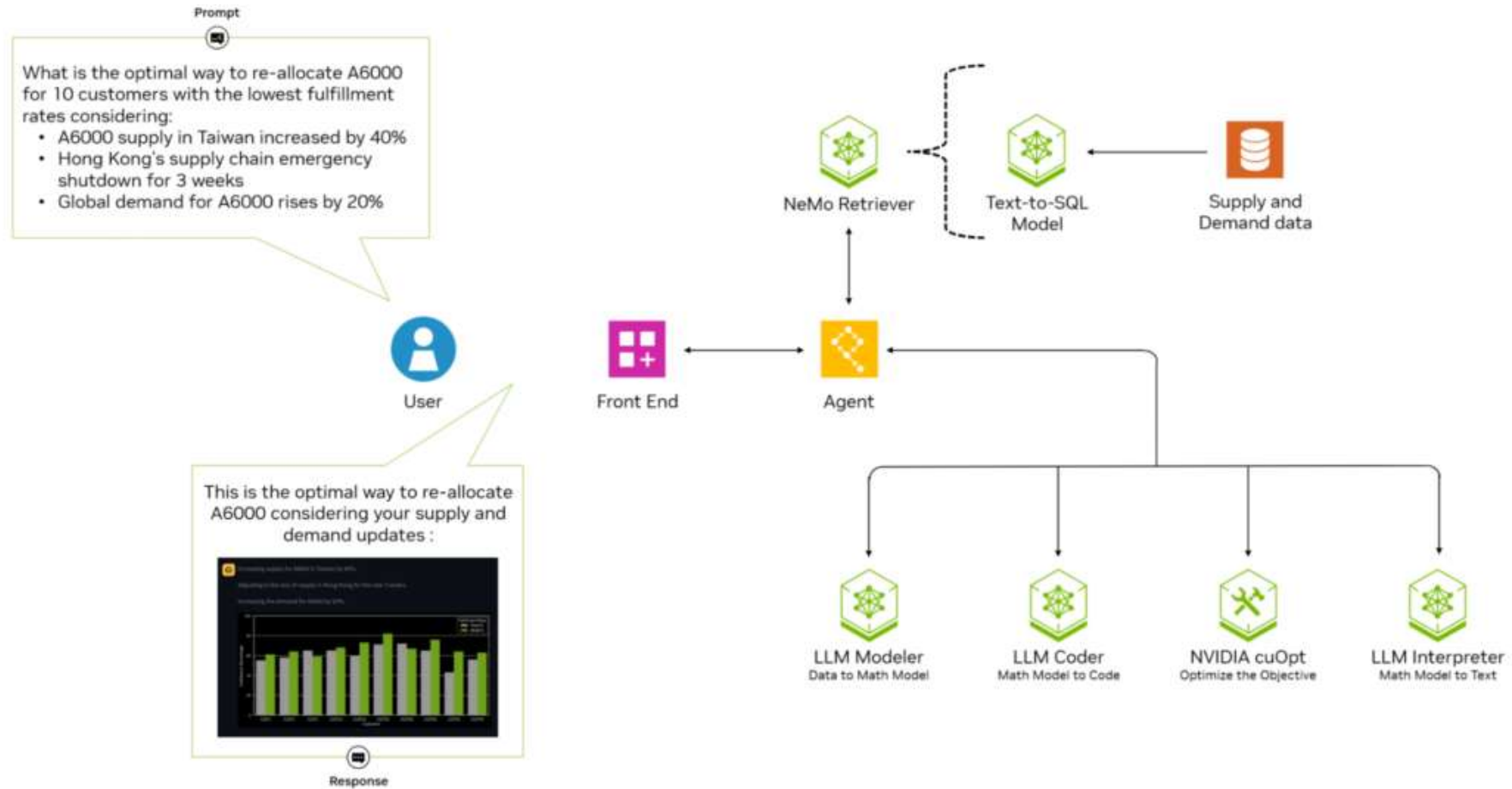


Figure 1: *OptiGuide* interface.

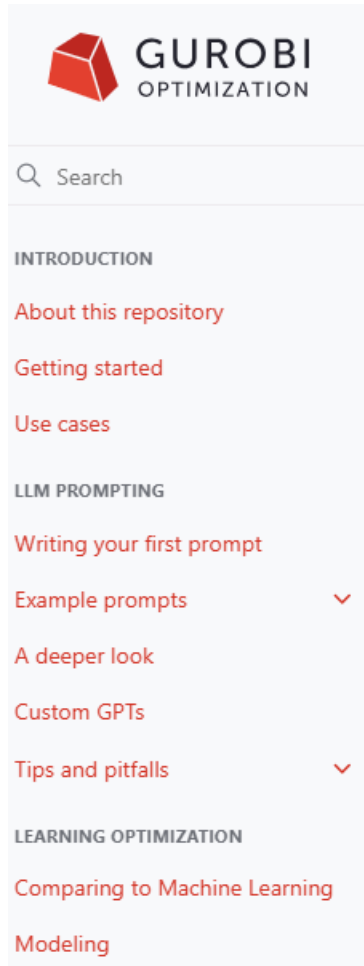


NVIDIA: supply chain agent



<https://developer.nvidia.com/blog/building-an-ai-agent-for-supply-chain-optimization-with-nvidia-nim-and-cuopt/>

Gurobi: modeling assistant



Welcome to Gurobi AI Modeling!

`gurobi-ai-modeling` aims to familiarize users with using generative AI for mathematical modeling. This repository contains documentation that features the following:

- Help users understand the capabilities and limitations of the state-of-the-art generative AI tools in the context of optimization.
- Propose best practices for using generative AI to build and solve optimization models and describe common pitfalls.
- Share examples of prompts and expected outcomes.
- Describe tools that can help create and solve models.

> Contents

INTRODUCTION

- [About this repository](#)
 - [How to utilize this repository](#)
 - [Structure](#)
 - [What to take away from this](#)
- [Getting started](#)
 - [Using ChatGPT as a translator for mathematics](#)
- [Use cases](#)
 - [Specialties](#)



ORTEC: assistant

The screenshot displays the ORTEC Assistant interface. On the left, a sidebar shows 'Territories (4)' with a table of territory assignments. The main area features a map of Amsterdam with customer locations marked by colored dots. On the right, a chat window titled 'ORTEC Assistant' is open, displaying a message about clustering and a response from the assistant.

Territory assignment

Name	Work time	Pal
Territory A	55 h 39 min	732
Territory Y	48 h 15 min	592
Territory C	58 h 43 min	712
Territory D	49 h 23 min	336

Customers (100)

Tags: ☒ All ☐ Unassigned ☐ Assigned

<input type="checkbox"/> Name	Territory	Tot. duration	Tot. pal	Location	Tags
Customer 1	-	1 h 0 min	20	Location 1	
Customer 2	Territory A	34 min	24	Location 2	
Customer 3	Territory Y	1 h 16 min	28	Location 3	
Customer 4	Territory A	40 min	20	Location 4	
Customer 5	Territory D	5 h 12 min	20	Location 5	

Total work time: 212 h 0 min

ORTEC Assistant

My solution is not clustered enough, what can we do about that?

Thanks for letting me know you're concerned about clustering! When a solution isn't "clustered enough," it usually means that customers in the same territory aren't located close to each other, or the territories aren't as geographically compact as you'd like.

Here are some ways we can address this:

- We can run territory optimization prioritizing clustering, so territories become more geographically focused—helping ensure customers in the same territory are closer together.
- This will aim to improve clustering, but there may be minor trade-offs, like a slightly less balanced workload or a small increase in travel time.

Would you like me to start an optimization that puts more emphasis on clustering to create more compact territories for you?

Message ORTEC Assistant

AI-generated content may be incorrect.

The use of Generative Artificial Intelligence (GenAI) in operations research: review and future research agenda

Qin Zhou^a and Jiuh-Biing Sheu^b

^aUniversity of Southampton Business School, Southampton, UK; ^bDepartment of Management Science, National Taiwan University, Taiwan, R.O.C

ABSTRACT

The emergence of Generative Artificial Intelligence (GenAI) represents a significant advancement in computational capabilities, offering transformative potential in Operations Research (OR). This study explores the role of GenAI through a systematic literature review. Following a careful analysis of the collected papers, they are classified into two main categories based on their nature: (1) application papers and (2) review and position papers. The latter provide a conceptual overview of GenAI's broader implications for OR, while the application papers are organized into a taxonomy encompassing three core dimensions: (1) GenAI for mathematical programming and optimization, (2) GenAI for stochastic systems, and (3) GenAI for simulation, strategic analysis, game theory, and risk management. Drawing insights from both conceptual and empirical studies, this review identifies cross-cutting themes and outlines a future research agenda to guide continued exploration at the intersection of GenAI and OR.

...ive data analysis to become active collaborators in the OR workflow.² GenAI is now being applied to tasks such as translating natural language³ into formal problem formulations, generating novel solution algorithms, synthesizing high-fidelity data for robust simulation, and modelling complex strategic interactions with unprecedented realism.⁴ Its unique strength lies in bridging the gap between the unstructured, human-centric world of language and the structured, formal domain of mathematical optimization, thereby enhancing the adaptability and scope of operational processes.⁵ F







Y

Software in the era of AI

Andrej Karpathy
Founding Member, OpenAI

Software

Software Operations Research in the era of AI

- LLMs = operating systems
 - We're in the 70s: terminal with 100k memory
- LLM psychology
 - Encyclopedic knowledge with severe amnesia
- Opportunities
 - Partial autonomy AI applications:
 - Context management
 - LLM orchestrations
 - Integrated GUI
 - Autonomy slider
 - Human-AI verification, vibe-coding, demo (`works.any()`) vs product (`works.all()`), build for agents



[Andrej Karpathy: Software Is Changing \(Again\)](#)

The Takeaway

“Real world problems are rarely as clean as our equations!”



