MIT Open Access Articles

Using Large Language Models for Evolutionary Search

The MIT Faculty has made this article openly available. ***Please share*** how this access benefits you. Your story matters.

Citation: OREILLY, UNA-MAY and Hemberg, Erik. 2024. "Using Large Language Models for Evolutionary Search."

As Published: https://doi.org/10.1145/3638530.3648432

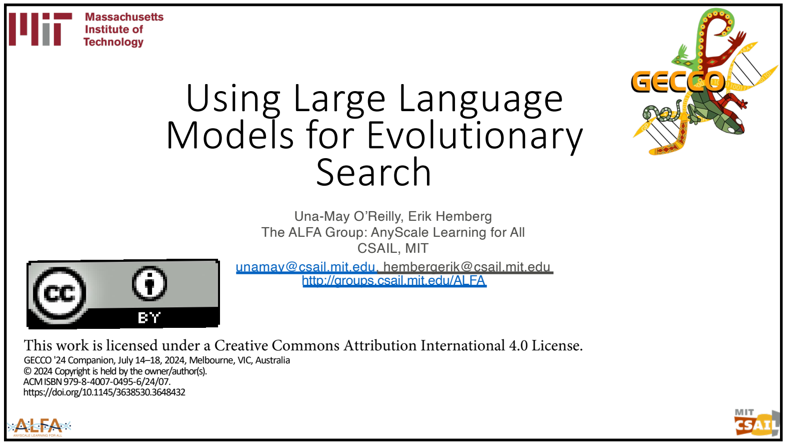
Publisher: ACM|Genetic and Evolutionary Computation Conference

Persistent URL: https://hdl.handle.net/1721.1/156670

Version: Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

Terms of use: Creative Commons Attribution

Massachusetts Institute of Technology



Massachusetts Institute of Technology

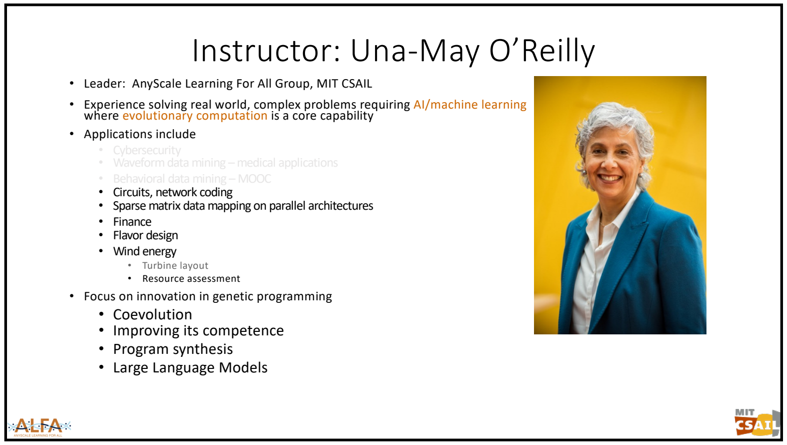
Using Large Language Models for Evolutionary Search

Una-May O'Reilly, Erik Hemberg
The ALFA Group: AnyScale Learning for All
CSAIL, MIT
unamay@csail.mit.edu, hembergerik@csail.mit.edu
http://groups.csail.mit.edu/ALFA

This work is licensed under a Creative Commons Attribution International 4.0 License.

GECCO '24 Companion, July 14–18, 2024, Melbourne, VIC, Australia
© 2024 Copyright is held by the owner/author(s).
ACM ISBN 979-8-4007-0495-6/24/07.
https://doi.org/10.1145/3638530.3648432

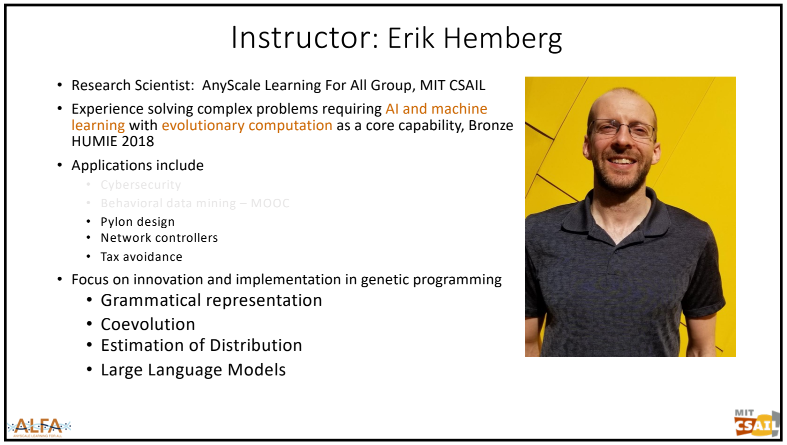
1



Instructor: Una-May O'Reilly

- Leader: AnyScale Learning For All Group, MIT CSAIL
- Experience solving real world, complex problems requiring AI/machine learning where evolutionary computation is a core capability
- Applications include
  - Cybersecurity
  - Waveform data mining – medical applications
  - Behavioral data mining – MOOC
  - Circuits, network coding
  - Sparse matrix data mapping on parallel architectures
  - Finance
  - Flavor design
  - Wind energy
    - Turbine layout
    - Resource assessment
- Focus on innovation in genetic programming
  - Coevolution
  - Improving its competence
  - Program synthesis
  - Large Language Models

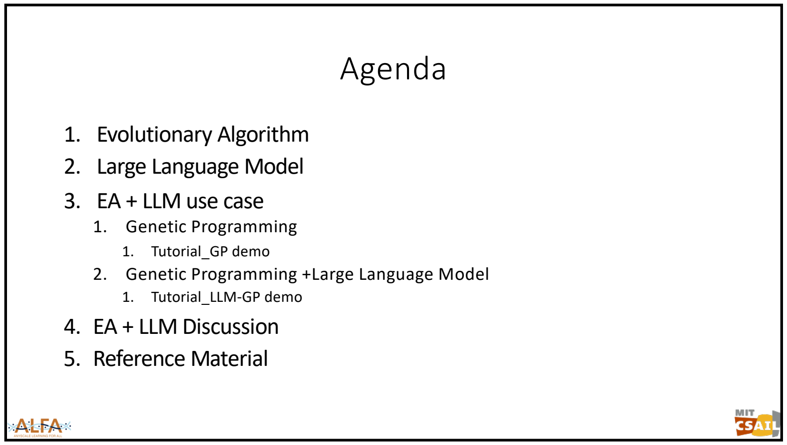
2



Instructor: Erik Hemberg

- Research Scientist: AnyScale Learning For All Group, MIT CSAIL
- Experience solving complex problems requiring AI and machine learning with evolutionary computation as a core capability, Bronze HUMIE 2018
- Applications include
  - Cybersecurity
  - Behavioral data mining – MOOC
  - Pylon design
  - Network controllers
  - Tax avoidance
- Focus on innovation and implementation in genetic programming
  - Grammatical representation
  - Coevolution
  - Estimation of Distribution
  - Large Language Models

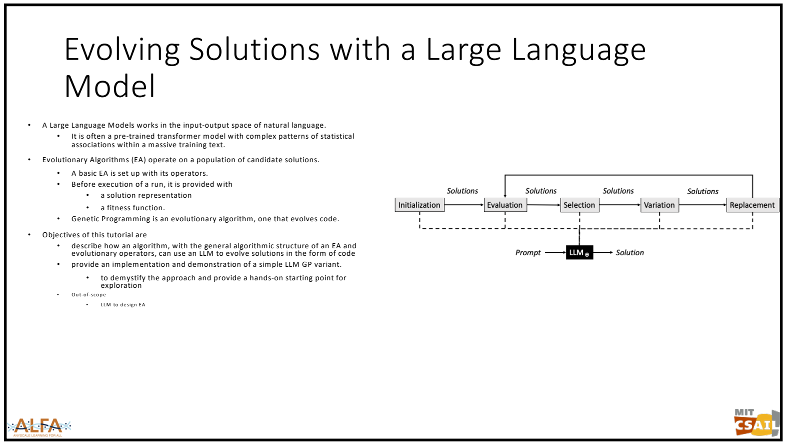
3



Agenda

1. Evolutionary Algorithm
2. Large Language Model
3. EA + LLM use case
   1. Genetic Programming
      1. Tutorial_GP demo
   2. Genetic Programming +Large Language Model
      1. Tutorial_LLM-GP demo
4. EA + LLM Discussion
5. Reference Material

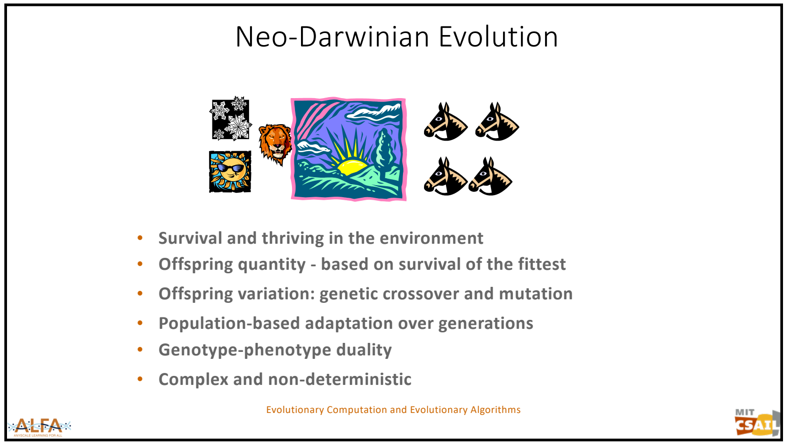
4



Evolving Solutions with a Large Language Model

- A Large Language Models works in the input-output space of natural language.
  - It is often a pre-trained transformer model with complex patterns of statistical associations within a massive training text.
- Evolutionary Algorithms (EA) operate on a population of candidate solutions.
  - A basic EA is set up with its operators.
  - Before execution of a run, it is provided with
    - a solution representation
    - a fitness function.
  - Genetic Programming is an evolutionary algorithm, one that evolves code.
- Objectives of this tutorial are
  - describe how an algorithm, with the general algorithmic structure of an EA and evolutionary operators, can use an LLM to evolve solutions in the form of code
  - provide an implementation and demonstration of a simple LLM GP variant.
    - to demystify the approach and provide a hands-on starting point for exploration
  - Out-of-scope
    - LLM to design EA

5

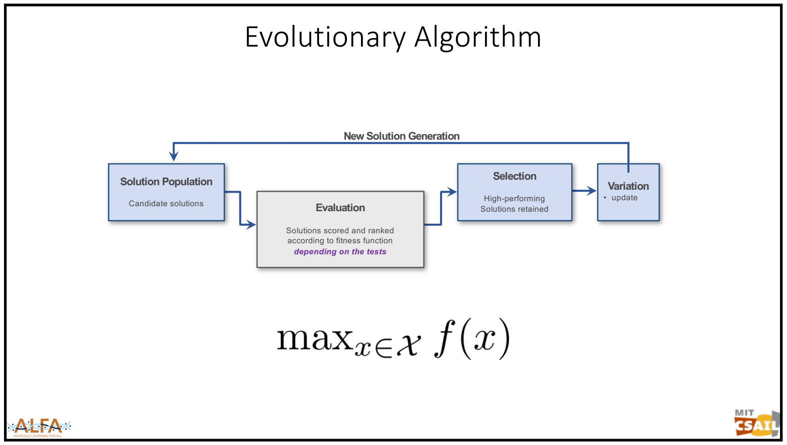


Neo-Darwinian Evolution

- Survival and thriving in the environment
- Offspring quantity - based on survival of the fittest
- Offspring variation: genetic crossover and mutation
- Population-based adaptation over generations
- Genotype-phenotype duality
- Complex and non-deterministic

Evolutionary Computation and Evolutionary Algorithms

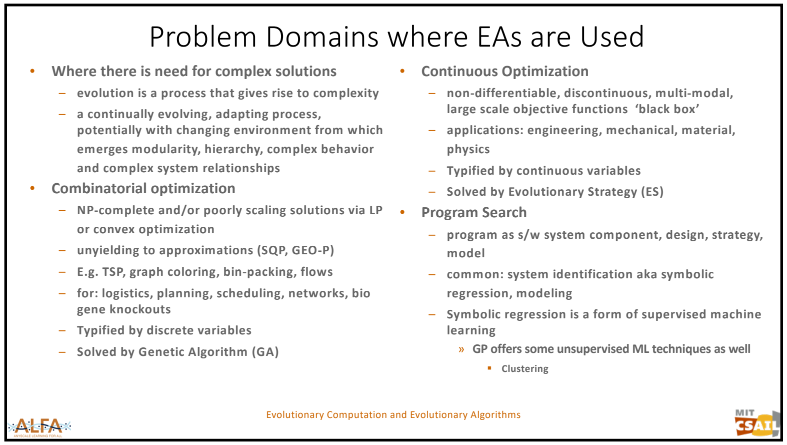
6



Evolutionary Algorithm

$$\max_{x \in \mathcal{X}} f(x)$$

7

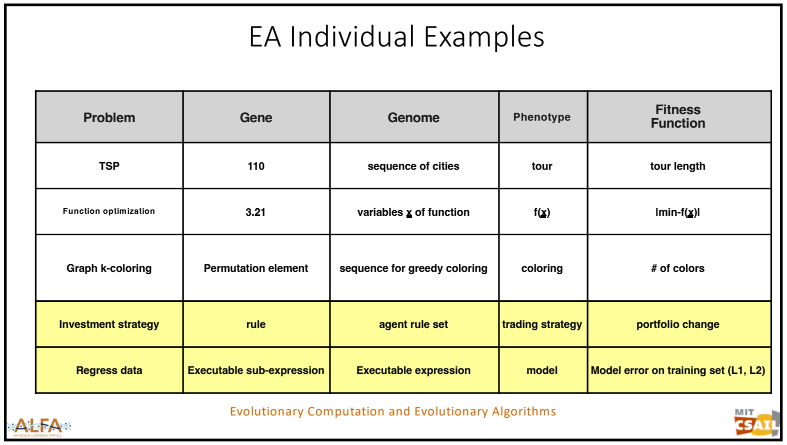


Problem Domains where EAs are Used

- Where there is need for complex solutions
  - evolution is a process that gives rise to complexity
  - a continually evolving, adapting process, potentially with changing environment from which emerges modularity, hierarchy, complex behavior and complex system relationships
- Combinatorial optimization
  - NP-complete and/or poorly scaling solutions via LP or convex optimization
  - unyielding to approximations (SQP, GEO-P)
  - E.g. TSP, graph coloring, bin-packing, flows
  - for: logistics, planning, scheduling, networks, bio gene knockouts
  - Typified by discrete variables
  - Solved by Genetic Algorithm (GA)
- Continuous Optimization
  - non-differentiable, discontinuous, multi-modal, large scale objective functions 'black box'
  - applications: engineering, mechanical, material, physics
  - Typified by continuous variables
  - Solved by Evolutionary Strategy (ES)
- Program Search
  - program as s/w system component, design, strategy, model
  - common: system identification aka symbolic regression, modeling
  - Symbolic regression is a form of supervised machine learning
    - » GP offers some unsupervised ML techniques as well
      - Clustering

Evolutionary Computation and Evolutionary Algorithms

8

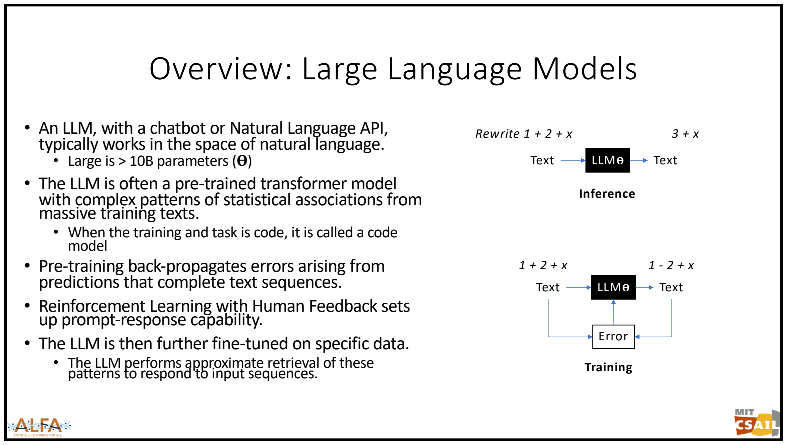


EA Individual Examples

Problem	Gene	Genome	Phenotype	Fitness Function
TSP	110	sequence of cities	tour	tour length
Function optimization	3.21	variables x of function	f(x)	|min-f(x)|
Graph k-coloring	Permutation element	sequence for greedy coloring	coloring	# of colors
Investment strategy	rule	agent rule set	trading strategy	portfolio change
Regress data	Executable sub-expression	Executable expression	model	Model error on training set (L1, L2)

Evolutionary Computation and Evolutionary Algorithms

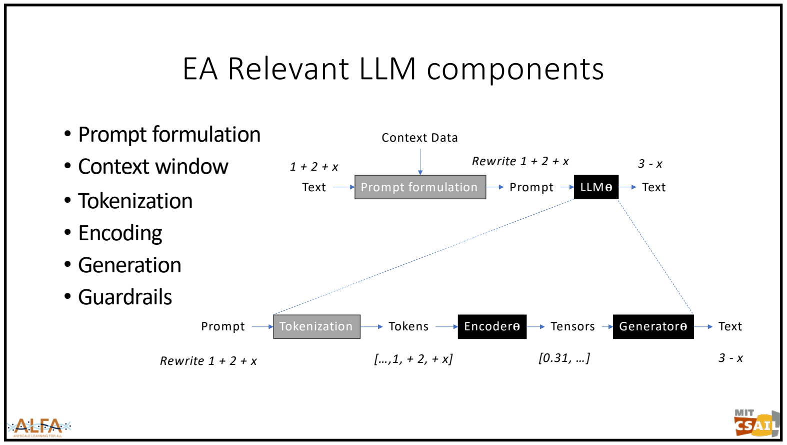
9



Overview: Large Language Models

- An LLM, with a chatbot or Natural Language API, typically works in the space of natural language.
  - Large is > 10B parameters (Θ)
- The LLM is often a pre-trained transformer model with complex patterns of statistical associations from massive training texts.
  - When the training and task is code, it is called a code model
- Pre-training back-propagates errors arising from predictions that complete text sequences.
- Reinforcement Learning with Human Feedback sets up prompt-response capability.
- The LLM is then further fine-tuned on specific data.
  - The LLM performs approximate retrieval of these patterns to respond to input sequences.

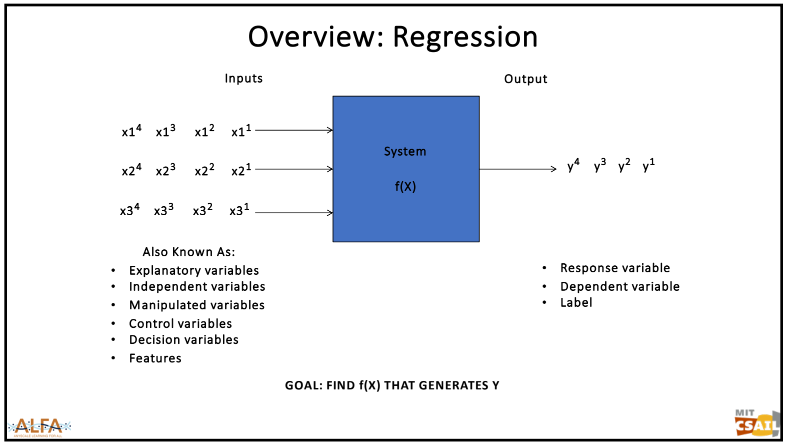
10



EA Relevant LLM components

- Prompt formulation
- Context window
- Tokenization
- Encoding
- Generation
- Guardrails

11



Overview: Regression

Also Known As:

- Explanatory variables
- Independent variables
- Manipulated variables
- Control variables
- Decision variables
- Features

- Response variable
- Dependent variable
- Label

GOAL: FIND f(X) THAT GENERATES Y

12

Regression

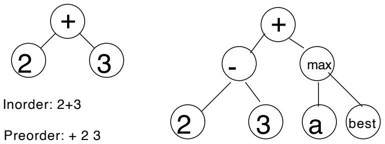
- Regress a relationship between a set of explanatory variables and a response variable
- Linear regression:
  - Assume linear model: y=ax+b
  - Optimize parameters (a,b) so data best fits model
- Logistic regression for classification
  - Maps linear model into sigmoid family

$$F(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

- Symbolic regression does NOT assume a model
  - Not parameter search
  - Model is intrinsic in GP solutions

13

Genetic Programming Parse Trees



Inorder: 2+3

Preorder: + 2 3

Post-order: 2 3 +

Inorder: (2-3) + (a max best)

Preorder: (+ (-2 3) (max a best))

Post-order: (2 3 -) (a best max) +)

Invariant to parse order:
- Preorder (node, left-child, right-child)
- Post-order (left-child, right-child, node)
- Inorder (left, node, right)

GP Evolves Executable Expressions

14

A Lisp GP system

A Lisp GP system is a large set of functions which are interpreted by evaluating the entry function

- Some are definitions of primitives you write!
  - (defun protectedDivide …)
- Rest is software logic for evolutionary algorithms

A GP system has functions that are pre-defined (by compilation or interpretation) for use as primitives and logic that handles

- Population initialization, iteration, selection, breeding, replacement, *fitness evalution*

GP expressions are first class objects in LISP so the GP software logic can manipulate them as data/variables as well as have the interpreter read and evaluate them

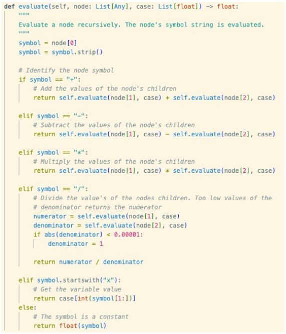
Expressions are data and are executed

GP Evolves Executable Expressions

15

How to Evaluate an Expression in GP

- interpreter beneath your code
  - Lisp example
- interpreter within your code
  - typical,
  - examples: tutorial_gp.py
- compile then execute on your OS

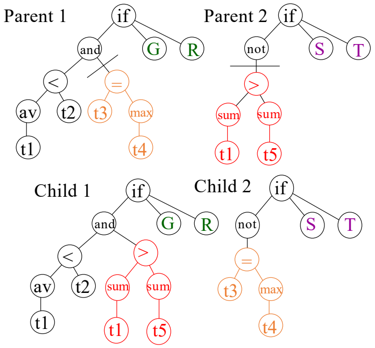


```
def evaluate(self, node: List[Any], case: List[float]) -> float:
    """
    Evaluate a node recursively. The node's symbol string is evaluated.
    """
    symbol = node[0]
    symbol = symbol.strip()

    # Identify the node symbol
    if symbol == "+":
        # Add the values of the node's children
        return self.evaluate(node[1], case) + self.evaluate(node[2], case)

    elif symbol == "-":
        # Subtract the values of the node's children
        return self.evaluate(node[1], case) - self.evaluate(node[2], case)

    elif symbol == "*":
        # Multiply the values of the node's children
        return self.evaluate(node[1], case) * self.evaluate(node[2], case)

    elif symbol == "/":
        # Divide the value's of the nodes children. Too low values of the
        # denominator returns the numerator
        numerator = self.evaluate(node[1], case)
        denominator = self.evaluate(node[2], case)
        if abs(denominator) < 0.00001:
            denominator = 1

        return numerator / denominator

    elif symbol.startswith("x"):
        # Get the variable value
        return case[int(symbol[1:])]
    else:
        # The symbol is a constant
        return float(symbol)
```

16

How to Manipulate and Vary Expressions as Data

- For Crossover and Mutation
  - offspring can be different size and structure than parents
  - syntactic correctness
  - randomness in replication and variation
- GP solution
  - reference the parse tree
  - Crossover - swap subtrees between trees of parents
  - Mutation: insert, substitute or delete from a parse tree (PT)



17

GP Preparatory Steps

Assume we have a GP system with internal expression evaluator.

1. Decide upon functions and terminals
   - Terminals bind to decision variables in problem
   - Combinatorial expression space defines the search space
2. Set up the fitness function
   - Translation of problem goal to GP goal
   - Minimization of error between desired and evolved expression when executed
   - Maximization of a problem-based score
   - Construct test cases for program (input examples, desired output)
3. Decide upon run parameters
   - Population size is most important
   - GP is robust to many other parameter choices
4. Determine a halt criteria and result to be returned
   - Maximum number of fitness evaluations
   - Time
   - Minimum acceptable error
   - Good enough solution (satisficing)

Nuts and Bolts GP Design

18

Top Level GP Algorithm

```
Begin
    pop = random programs from a set of operators and operands
    repeat
        execute each program in pop with each set of inputs
        measure each program's fitness
        repeat
            select 2 parents
            copy 2 offspring from parents
                crossover
                mutate
            add to new-pop
        until pop-size
    pop = new-pop
    until max-generation
        or
         adequate program found
End
```

Grow or Full

Ramped-half-half

Max-init-tree-height

Prepare input data
Designate solution
Define error between actual and expected

- Tournament selection
- Fitness proportional selection

Tournament size

- Subtree substitution
- Permute
- Edit

Mutation probs

Sub-tree crossover

Prob to crossover

Max-tree-height

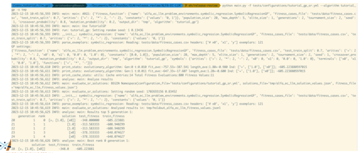
Leaf:node bias

Nuts and Bolts GP Design - Summary

19

Tutorial_GP: Simple Symbolic Regression

- Given a set of independent decision variables and corresponding values for a dependent variable
- Want: a model that predicts the dependent variable
  - Eg: linear model with numerical coefficients
    - $Y= aX_1 + bX_2 + c(X_1X_2)$
  - Eg: non-linear model
    - $y= a X_1^2 + bX_2^3$
  - Prediction accuracy: minimum error between model prediction and actual samples
- Usually: designer provides a model, and a regression (ordinary least squares, Fourier series) determines coefficients
- With genetic programming, the model (structure) and the coefficients can be learned



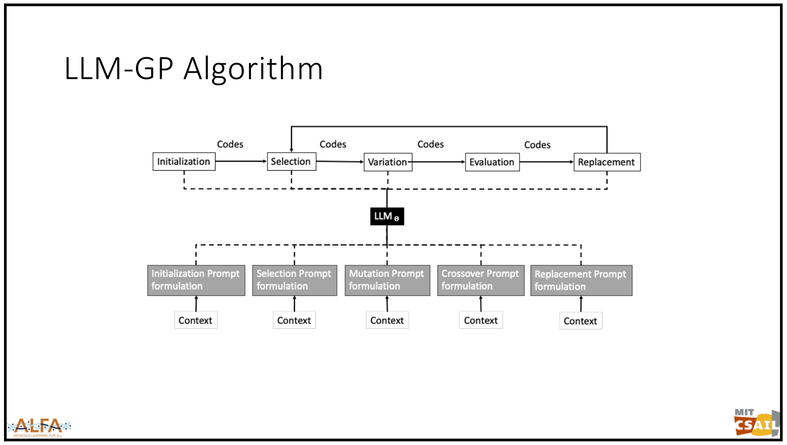
```
(venv_tutorial_llm_gp) erikhemberg@beech ~/Documents/MIT_PostDoc/LLM/release_review/ALFA-EC-LLM ? eh/release-review python main.py -f tests/configurations/tutorial_gp_sr.yml --algorithm tutorial_
gp -o tmp
2023-12-15 10:45:56,593 INFO: main: main: ARGS: {'fitness_function': {'name': 'alfa_ec_llm.problem_environments.symbolic_regression.SymbolicRegressionGP', 'fitness_cases_file': 'tests/data/fitness_cases.c
sv', 'test_train_split': 0.7, 'arities': {'+': 2, '*': 2, '-': 2}, 'constants': {'values': '0, 1'}}, 'population_size': 20, 'max_depth': 5, 'elite_size': 1, 'generations': 2, 'tournament_size': 2, 'seed':
 1, 'crossover_probability': 0.8, 'mutation_probability': 0.2, 'output_dir': 'tmp', 'algorithm': 'tutorial_gp'}
2023-12-15 10:45:56,593 INFO: main: main: Use TutorialGP
2023-12-15 10:45:56,593 INFO: run: tutorial_gp: Setting random seed: 1 0.13436
2023-12-15 10:45:56,593 INFO: __init__: symbolic_regression: {'name': 'alfa_ec_llm.problem_environments.symbolic_regression.SymbolicRegressionGP', 'fitness_cases_file': 'tests/data/fitness_cases.csv', 'te
st_train_split': 0.7, 'arities': {'+': 2, '*': 2, '-': 2}, 'constants': {'values': '0, 1'}}
2023-12-15 10:45:56,593 INFO: parse_exemplars: symbolic_regression: Reading: tests/data/fitness_cases.csv headers: ['# x0', 'x1', 'y'] exemplars: 121
GP settings:
{'fitness_function': {'name': 'alfa_ec_llm.problem_environments.symbolic_regression.SymbolicRegressionGP', 'fitness_cases_file': 'tests/data/fitness_cases.csv', 'test_train_split': 0.7, 'arities': {'+': 2
, '*': 2, '-': 2, 'x0': 0, 'x1': 0, '0.0': 0, '1.0': 0}, 'constants': [0.0, 1.0]}, 'population_size': 20, 'max_depth': 5, 'elite_size': 1, 'generations': 2, 'tournament_size': 2, 'seed': 1, 'crossover_pro
bability': 0.8, 'mutation_probability': 0.2, 'output_dir': 'tmp', 'algorithm': 'tutorial_gp', 'symbols': {'arities': {'+': 2, '*': 2, '-': 2, 'x0': 0, 'x1': 0, '0.0': 0, '1.0': 0}, 'terminals': ['x0', 'x1
', '0.0', '1.0'], 'functions': ['+', '*', '-']}}
2023-12-15 10:45:56,610 INFO: print_stats: evolutionary_algorithm: Gen:0 t:0.010 fit_ave:-757.53+-387.541 length_ave:1.80+-0.980 Ind: ['+', ['1.0'], ['x0']]; -605.223880597015
2023-12-15 10:45:56,611 INFO: print_stats: evolutionary_algorithm: Gen:1 t:0.011 fit_ave:-647.33+-17.607 length_ave:1.20+-0.600 Ind: ['+', ['1.0'], ['x0']]; -605.223880597015
2023-12-15 10:45:56,611 INFO: print_cache_stats: utils: Cache entries:14 Total Fitness Evaluations:800 Fitness Values:11
2023-12-15 10:45:56,612 INFO: analyse: main: Analyse results
2023-12-15 10:45:56,612 INFO: main: evaluate_sr_solutions: BEGIN Namespace(configuration_file='tests/configurations/tutorial_gp_sr.yml', solutions_file='tmp/alfa_ec_llm_solution_values.json', fitness_file
='tmp/alfa_ec_llm_fitness_values.json')
2023-12-15 10:45:56,613 INFO: main: evaluate_sr_solutions: Setting random seed: 1702655156 0.83452
2023-12-15 10:45:56,613 INFO: __init__: symbolic_regression: {'name': 'alfa_ec_llm.problem_environments.symbolic_regression.SymbolicRegressionGP', 'fitness_cases_file': 'tests/data/fitness_cases.csv', 'te
st_train_split': 0.7, 'arities': {'+': 2, '*': 2, '-': 2}, 'constants': {'values': '0, 1'}}
2023-12-15 10:45:56,614 INFO: parse_exemplars: symbolic_regression: Reading: tests/data/fitness_cases.csv headers: ['# x0', 'x1', 'y'] exemplars: 121
2023-12-15 10:45:56,619 INFO: main: evaluate_sr_solutions: Analysed results in: tmp/holdout_alfa_ec_llm_fitness_values.jsonl
2023-12-15 10:45:56,625 INFO: analyse: main: Results top 5 generation 1:
    generation  rank          solution  test_fitness  train_fitness
20           1     0  [+, [1.0], [x0]]   -348.000000    -605.223881
21           1     1             [1.0]   -313.583333    -608.940299
22           1     2             [1.0]   -313.583333    -608.940299
23           1     3              [x0]   -378.333333    -648.074627
24           1     4              [x0]   -378.333333    -648.074627
2023-12-15 10:45:56,626 INFO: analyse: main: Best rank 0 generation 1:
            solution  test_fitness  train_fitness
20  [+, [1.0], [x0]]        -348.0    -605.223881
```

- Test problem:
  - f(x)=(X_0 * X_0) + (X_1 * X_1)
- Domain of X_0 and X_1 [-5.0,5.0]
- Choose the 4 operands (terminals)
  - X_0, X_1, 1.0, 0
- Choose the 4 operators (functions)
  - +, - , *, / (protected)
  - protected divide: if denominator==0, return numerator
- Fitness function: sum of mean squared error between y_i, and expression's return values
- Prepare 121 randomized points for testing
- Out of sample training:testing ratio is 70:30, random selection of points as training or test

Examples

GOTO: VS Code debugger
- Evaluation
- Mutation Operator

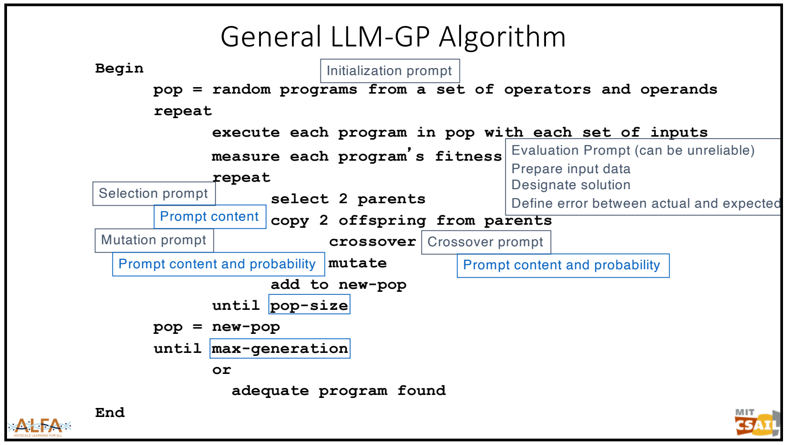
20



LLM-GP Algorithm

Initialization
Codes
Selection
Codes
Variation
Codes
Evaluation
Codes
Replacement
LLM θ
Initialization Prompt formulation
Selection Prompt formulation
Mutation Prompt formulation
Crossover Prompt formulation
Replacement Prompt formulation
Context
Context
Context
Context
Context

21



General LLM-GP Algorithm

```
Begin
        pop = random programs from a set of operators and operands
        repeat
                execute each program in pop with each set of inputs
                measure each program's fitness
                repeat
                        select 2 parents
                        copy 2 offspring from parents
                                crossover
                                mutate
                        add to new-pop
                until pop-size
        pop = new-pop
        until max-generation
                or
                  adequate program found
End
```

Initialization prompt

Evaluation Prompt (can be unreliable)
Prepare input data
Designate solution
Define error between actual and expected

Selection prompt

Prompt content

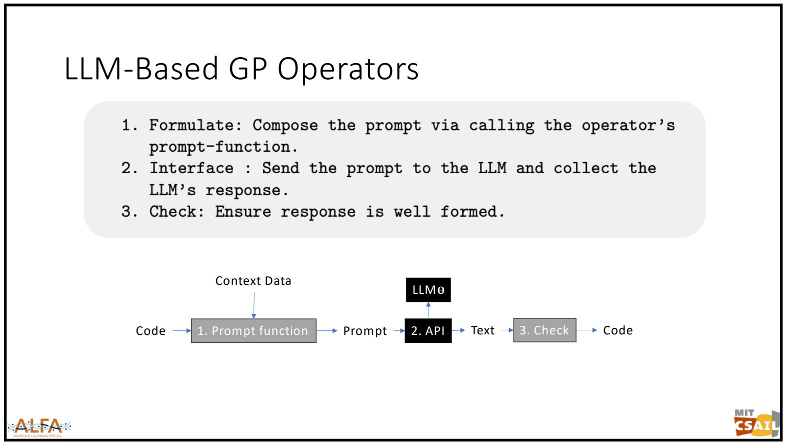
Mutation prompt

Prompt content and probability

Crossover prompt

Prompt content and probability

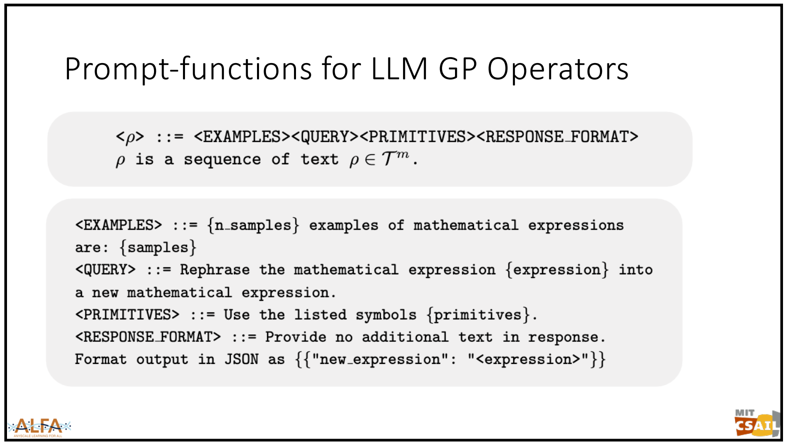
22



LLM-Based GP Operators

1. Formulate: Compose the prompt via calling the operator's prompt-function.
2. Interface : Send the prompt to the LLM and collect the LLM's response.
3. Check: Ensure response is well formed.

23

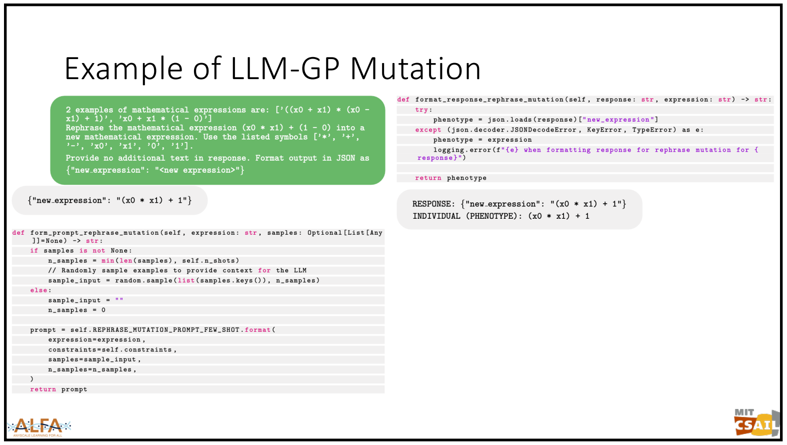


Prompt-functions for LLM GP Operators

<ρ> ::= <EXAMPLES><QUERY><PRIMITIVES><RESPONSE_FORMAT>
ρ is a sequence of text $\rho \in \mathcal{T}^m$.

<EXAMPLES> ::= {n_samples} examples of mathematical expressions are: {samples}
<QUERY> ::= Rephrase the mathematical expression {expression} into a new mathematical expression.
<PRIMITIVES> ::= Use the listed symbols {primitives}.
<RESPONSE_FORMAT> ::= Provide no additional text in response. Format output in JSON as {{"new_expression": "<expression>"}}

24



Example of LLM-GP Mutation

```
2 examples of mathematical expressions are: ['((x0 + x1) * (x0 -
x1) + 1)', 'x0 + x1 * (1 - 0)']
Rephrase the mathematical expression (x0 * x1) + (1 - 0) into a
new mathematical expression. Use the listed symbols ['*', '+',
'-', 'x0', 'x1', '0', '1'].
Provide no additional text in response. Format output in JSON as
{"new_expression": "<new expression>"}
```

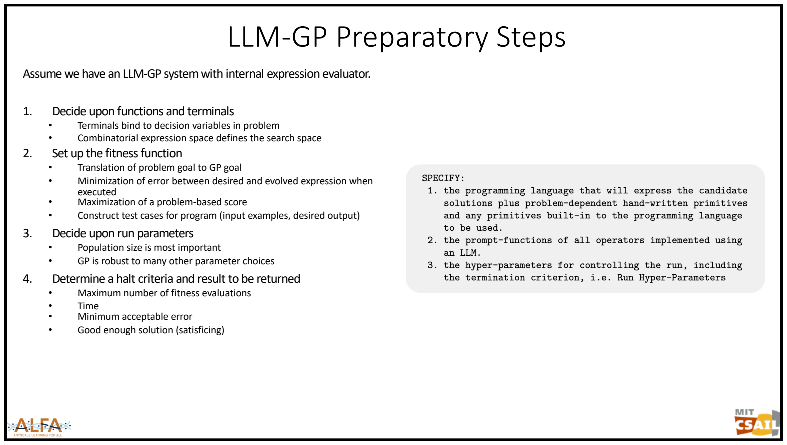
```
{"new_expression": "(x0 * x1) + 1"}
```

```
def form_prompt_rephrase_mutation(self, expression: str, samples: Optional[List[Any
    ]]=None) -> str:
    if samples is not None:
        n_samples = min(len(samples), self.n_shots)
        // Randomly sample examples to provide context for the LLM
        sample_input = random.sample(list(samples.keys()), n_samples)
    else:
        sample_input = ""
        n_samples = 0

    prompt = self.REPHRASE_MUTATION_PROMPT_FEW_SHOT.format(
        expression=expression,
        constraints=self.constraints,
        samples=sample_input,
        n_samples=n_samples,
    )
    return prompt
```

```
def format_response_rephrase_mutation(self, response: str, expression: str) -> str:
    try:
        phenotype = json.loads(response)["new_expression"]
    except (json.decoder.JSONDecodeError, KeyError, TypeError) as e:
        phenotype = expression
        logging.error(f"{e} when formatting response for rephrase mutation for {
    response}")

    return phenotype
```

```
RESPONSE: {"new_expression": "(x0 * x1) + 1"}
INDIVIDUAL (PHENOTYPE): (x0 * x1) + 1
```

25



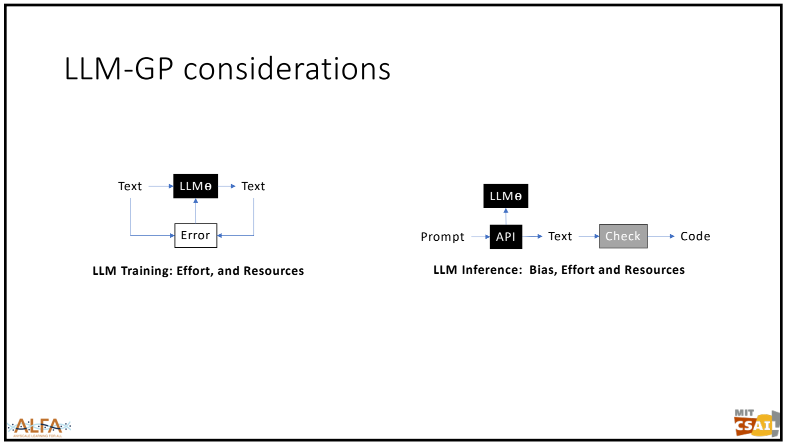
LLM-GP Preparatory Steps

Assume we have an LLM-GP system with internal expression evaluator.

1. Decide upon functions and terminals
   - Terminals bind to decision variables in problem
   - Combinatorial expression space defines the search space
2. Set up the fitness function
   - Translation of problem goal to GP goal
   - Minimization of error between desired and evolved expression when executed
   - Maximization of a problem-based score
   - Construct test cases for program (input examples, desired output)
3. Decide upon run parameters
   - Population size is most important
   - GP is robust to many other parameter choices
4. Determine a halt criteria and result to be returned
   - Maximum number of fitness evaluations
   - Time
   - Minimum acceptable error
   - Good enough solution (satisficing)

```
SPECIFY:
 1. the programming language that will express the candidate
    solutions plus problem-dependent hand-written primitives
    and any primitives built-in to the programming language
    to be used.
 2. the prompt-functions of all operators implemented using
    an LLM.
 3. the hyper-parameters for controlling the run, including
    the termination criterion, i.e. Run Hyper-Parameters
```

26



LLM-GP considerations

Text → LLMθ → Text

Error

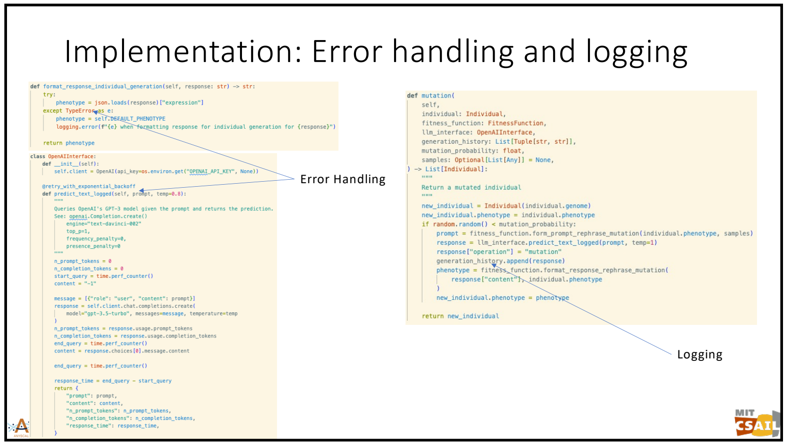
LLM Training: Effort, and Resources

LLMθ

Prompt → API → Text → Check → Code

LLM Inference: Bias, Effort and Resources

27



Implementation: Error handling and logging

```
def format_response_individual_generation(self, response: str) -> str:
    try:
        phenotype = json.loads(response)["expression"]
    except TypeError as e:
        phenotype = self.DEFAULT_PHENOTYPE
        logging.error(f"{e} when formatting response for individual generation for {response}")

    return phenotype
```

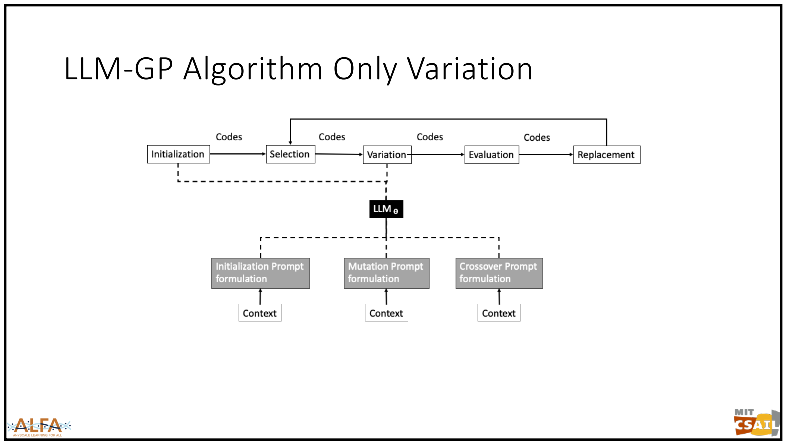
```
class OpenAIInterface:
    def __init__(self):
        self.client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY", None))

    @retry_with_exponential_backoff
    def predict_text_logged(self, prompt, temp=0.8):
        """
        Queries OpenAI's GPT-3 model given the prompt and returns the prediction.
        See: openai.Completion.create()
            engine="text-davinci-002"
            top_p=1,
            frequency_penalty=0,
            presence_penalty=0
        """
        n_prompt_tokens = 0
        n_completion_tokens = 0
        start_query = time.perf_counter()
        content = "-1"

        message = [{"role": "user", "content": prompt}]
        response = self.client.chat.completions.create(
            model="gpt-3.5-turbo", messages=message, temperature=temp
        )
        n_prompt_tokens = response.usage.prompt_tokens
        n_completion_tokens = response.usage.completion_tokens
        end_query = time.perf_counter()
        content = response.choices[0].message.content

        end_query = time.perf_counter()

        response_time = end_query - start_query
        return {
            "prompt": prompt,
            "content": content,
            "n_prompt_tokens": n_prompt_tokens,
            "n_completion_tokens": n_completion_tokens,
            "response_time": response_time,
        }
```

Error Handling

```
def mutation(
    self,
    individual: Individual,
    fitness_function: FitnessFunction,
    llm_interface: OpenAIInterface,
    generation_history: List[Tuple[str, str]],
    mutation_probability: float,
    samples: Optional[List[Any]] = None,
) -> List[Individual]:
    """
    Return a mutated individual
    """
    new_individual = Individual(individual.genome)
    new_individual.phenotype = individual.phenotype
    if random.random() < mutation_probability:
        prompt = fitness_function.form_prompt_rephrase_mutation(individual.phenotype, samples)
        response = llm_interface.predict_text_logged(prompt, temp=1)
        response["operation"] = "mutation"
        generation_history.append(response)
        phenotype = fitness_function.format_response_rephrase_mutation(
            response["content"], individual.phenotype
        )
        new_individual.phenotype = phenotype

    return new_individual
```

Logging

28



LLM-GP Algorithm Only Variation

Initialization → Codes → Selection → Codes → Variation → Codes → Evaluation → Codes → Replacement

LLM θ

Initialization Prompt formulation

Mutation Prompt formulation

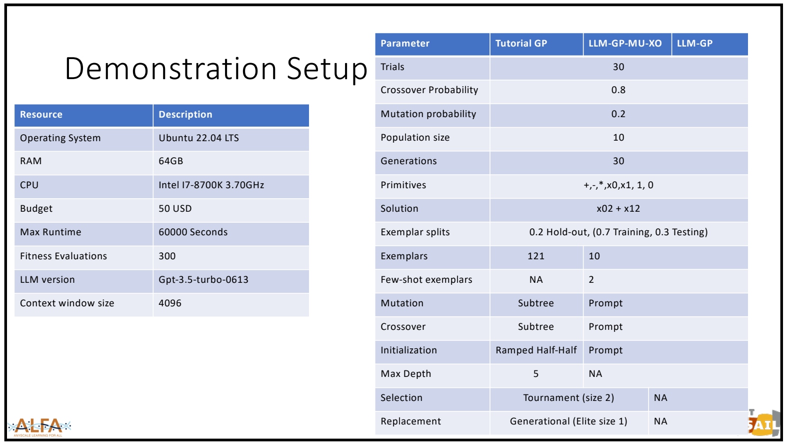
Crossover Prompt formulation

Context

Context

Context

29

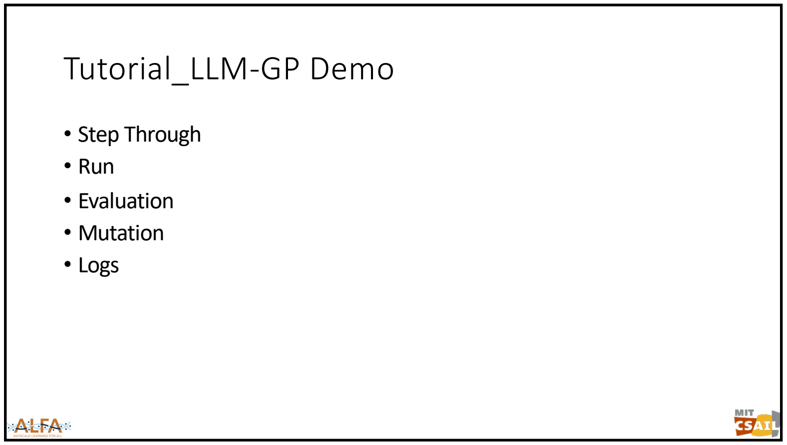


Demonstration Setup

Resource	Description
Operating System	Ubuntu 22.04 LTS
RAM	64GB
CPU	Intel I7-8700K 3.70GHz
Budget	50 USD
Max Runtime	60000 Seconds
Fitness Evaluations	300
LLM version	Gpt-3.5-turbo-0613
Context window size	4096

Parameter	Tutorial GP	LLM-GP-MU-XO	LLM-GP
Trials	30		
Crossover Probability	0.8		
Mutation probability	0.2		
Population size	10		
Generations	30		
Primitives	+,-,*,x0,x1, 1, 0		
Solution	x02 + x12		
Exemplar splits	0.2 Hold-out, (0.7 Training, 0.3 Testing)		
Exemplars	121	10	
Few-shot exemplars	NA	2	
Mutation	Subtree	Prompt	
Crossover	Subtree	Prompt	
Initialization	Ramped Half-Half	Prompt	
Max Depth	5	NA	
Selection	Tournament (size 2)		NA
Replacement	Generational (Elite size 1)		NA

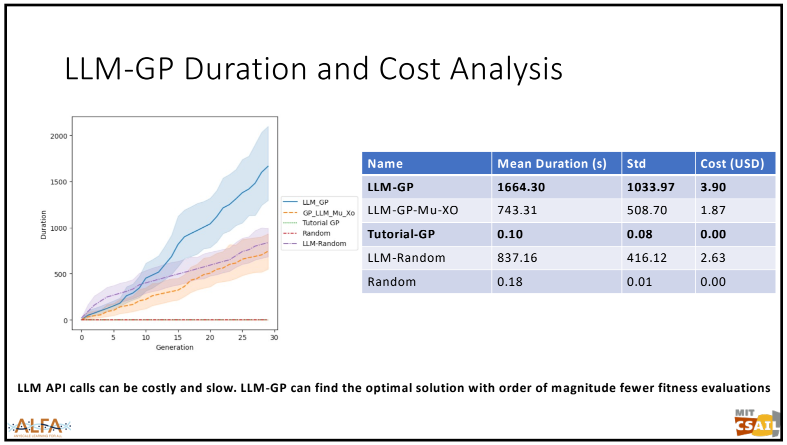
30



Tutorial_LLM-GP Demo

- Step Through
- Run
- Evaluation
- Mutation
- Logs

31



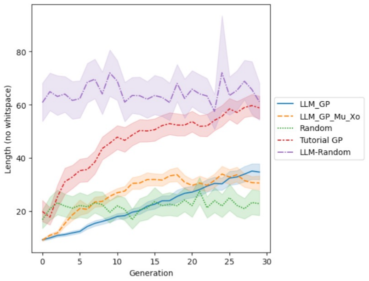
LLM-GP Duration and Cost Analysis

Name	Mean Duration (s)	Std	Cost (USD)
LLM-GP	**1664.30**	**1033.97**	**3.90**
LLM-GP-Mu-XO	743.31	508.70	1.87
Tutorial-GP	**0.10**	**0.08**	**0.00**
LLM-Random	837.16	416.12	2.63
Random	0.18	0.01	0.00

LLM API calls can be costly and slow. LLM-GP can find the optimal solution with order of magnitude fewer fitness evaluations

32

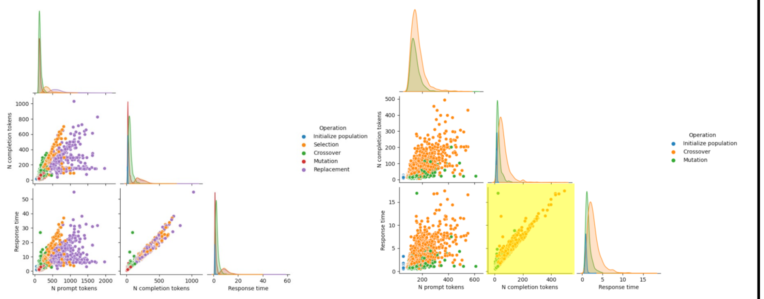
Size Analysis



LLM-GP solutions do not increase in size as much as Tutorial GP.
Note, there was no explicit solution size bias for any algorithm

33

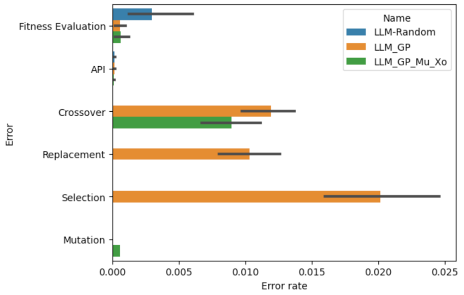
LLM use analysis



There might be a response time limit.

34

LLM-GP Error Analysis



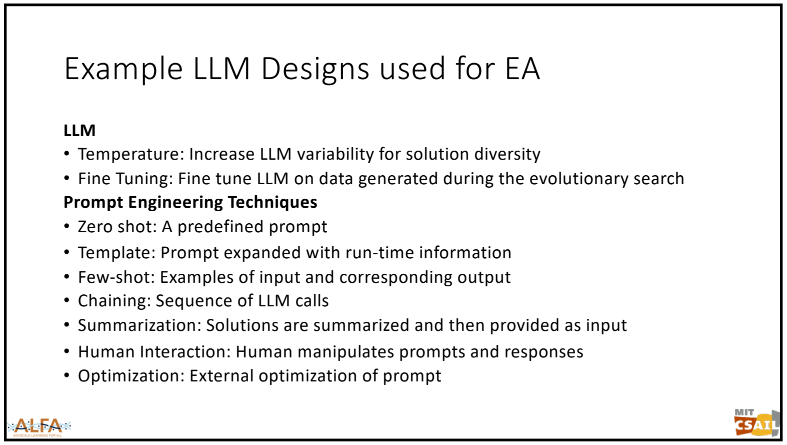
There is a distinct difference in LLM operator error rates

35

LLM-EA Example Domains

- Program synthesis
- Code for Agent Controller
- Boolean Parity
- Symbolic Regression (Function Search)
- Optimization Heuristics
- Neural Architecture Search
- Prompts
- Data for LLM Tuning

36



Example LLM Designs used for EA

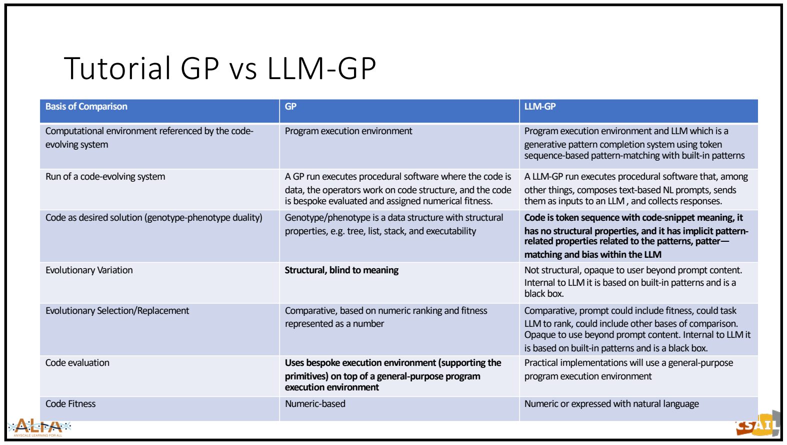
LLM

- Temperature: Increase LLM variability for solution diversity
- Fine Tuning: Fine tune LLM on data generated during the evolutionary search

Prompt Engineering Techniques

- Zero shot: A predefined prompt
- Template: Prompt expanded with run-time information
- Few-shot: Examples of input and corresponding output
- Chaining: Sequence of LLM calls
- Summarization: Solutions are summarized and then provided as input
- Human Interaction: Human manipulates prompts and responses
- Optimization: External optimization of prompt

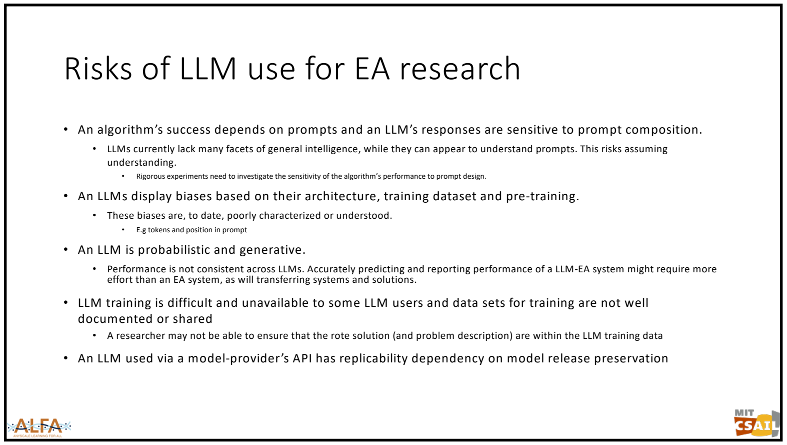
37



Tutorial GP vs LLM-GP

Basis of Comparison	GP	LLM-GP
Computational environment referenced by the code-evolving system	Program execution environment	Program execution environment and LLM which is a generative pattern completion system using token sequence-based pattern-matching with built-in patterns
Run of a code-evolving system	A GP run executes procedural software where the code is data, the operators work on code structure, and the code is bespoke evaluated and assigned numerical fitness.	A LLM-GP run executes procedural software that, among other things, composes text-based NL prompts, sends them as inputs to an LLM , and collects responses.
Code as desired solution (genotype-phenotype duality)	Genotype/phenotype is a data structure with structural properties, e.g. tree, list, stack, and executability	**Code is token sequence with code-snippet meaning, it has no structural properties, and it has implicit pattern-related properties related to the patterns, patter—matching and bias within the LLM**
Evolutionary Variation	**Structural, blind to meaning**	Not structural, opaque to user beyond prompt content. Internal to LLM it is based on built-in patterns and is a black box.
Evolutionary Selection/Replacement	Comparative, based on numeric ranking and fitness represented as a number	Comparative, prompt could include fitness, could task LLM to rank, could include other bases of comparison. Opaque to use beyond prompt content. Internal to LLM it is based on built-in patterns and is a black box.
Code evaluation	**Uses bespoke execution environment (supporting the primitives) on top of a general-purpose program execution environment**	Practical implementations will use a general-purpose program execution environment
Code Fitness	Numeric-based	Numeric or expressed with natural language

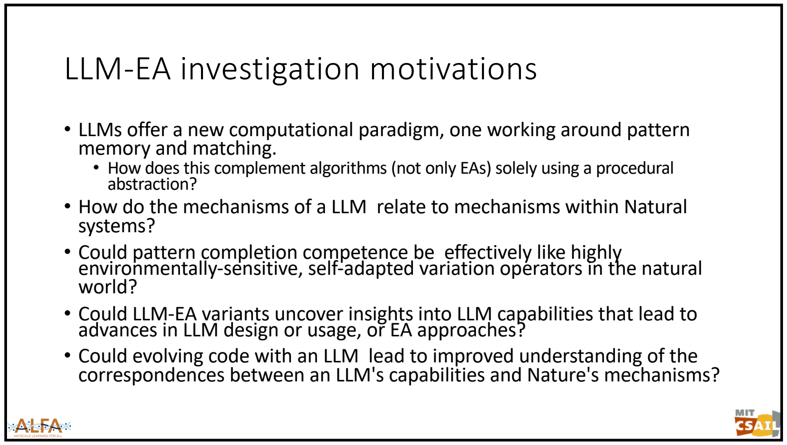
38



Risks of LLM use for EA research

- An algorithm's success depends on prompts and an LLM's responses are sensitive to prompt composition.
  - LLMs currently lack many facets of general intelligence, while they can appear to understand prompts. This risks assuming understanding.
    - Rigorous experiments need to investigate the sensitivity of the algorithm's performance to prompt design.
- An LLMs display biases based on their architecture, training dataset and pre-training.
  - These biases are, to date, poorly characterized or understood.
    - E.g tokens and position in prompt
- An LLM is probabilistic and generative.
  - Performance is not consistent across LLMs. Accurately predicting and reporting performance of a LLM-EA system might require more effort than an EA system, as will transferring systems and solutions.
- LLM training is difficult and unavailable to some LLM users and data sets for training are not well documented or shared
  - A researcher may not be able to ensure that the rote solution (and problem description) are within the LLM training data
- An LLM used via a model-provider's API has replicability dependency on model release preservation

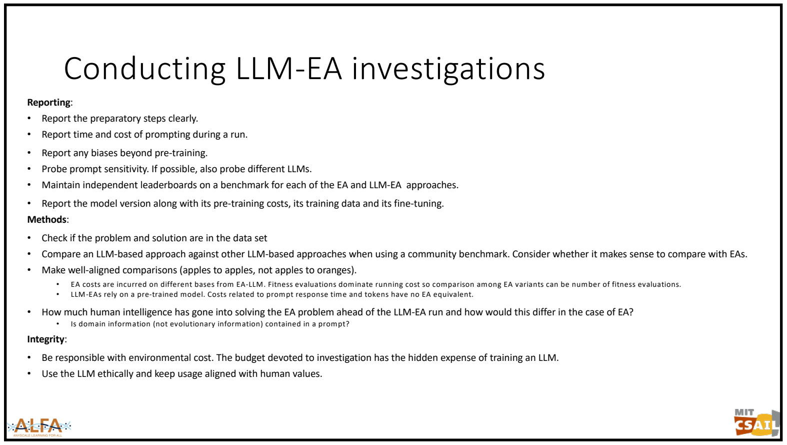
39



LLM-EA investigation motivations

- LLMs offer a new computational paradigm, one working around pattern memory and matching.
  - How does this complement algorithms (not only EAs) solely using a procedural abstraction?
- How do the mechanisms of a LLM relate to mechanisms within Natural systems?
- Could pattern completion competence be effectively like highly environmentally-sensitive, self-adapted variation operators in the natural world?
- Could LLM-EA variants uncover insights into LLM capabilities that lead to advances in LLM design or usage, or EA approaches?
- Could evolving code with an LLM lead to improved understanding of the correspondences between an LLM's capabilities and Nature's mechanisms?

40



Conducting LLM-EA investigations

Reporting:

- Report the preparatory steps clearly.
- Report time and cost of prompting during a run.
- Report any biases beyond pre-training.
- Probe prompt sensitivity. If possible, also probe different LLMs.
- Maintain independent leaderboards on a benchmark for each of the EA and LLM-EA approaches.
- Report the model version along with its pre-training costs, its training data and its fine-tuning.

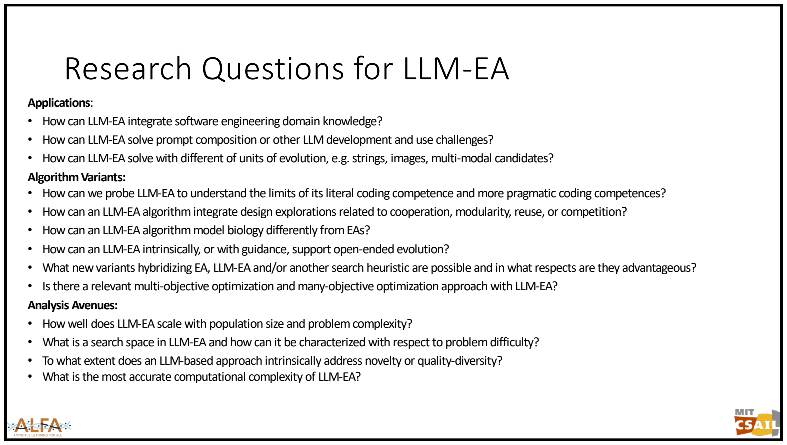
Methods:

- Check if the problem and solution are in the data set
- Compare an LLM-based approach against other LLM-based approaches when using a community benchmark. Consider whether it makes sense to compare with EAs.
- Make well-aligned comparisons (apples to apples, not apples to oranges).
    - EA costs are incurred on different bases from EA-LLM. Fitness evaluations dominate running cost so comparison among EA variants can be number of fitness evaluations.
    - LLM-EAs rely on a pre-trained model. Costs related to prompt response time and tokens have no EA equivalent.
- How much human intelligence has gone into solving the EA problem ahead of the LLM-EA run and how would this differ in the case of EA?
    - Is domain information (not evolutionary information) contained in a prompt?

Integrity:

- Be responsible with environmental cost. The budget devoted to investigation has the hidden expense of training an LLM.
- Use the LLM ethically and keep usage aligned with human values.

41



Research Questions for LLM-EA

Applications:

- How can LLM-EA integrate software engineering domain knowledge?
- How can LLM-EA solve prompt composition or other LLM development and use challenges?
- How can LLM-EA solve with different of units of evolution, e.g. strings, images, multi-modal candidates?

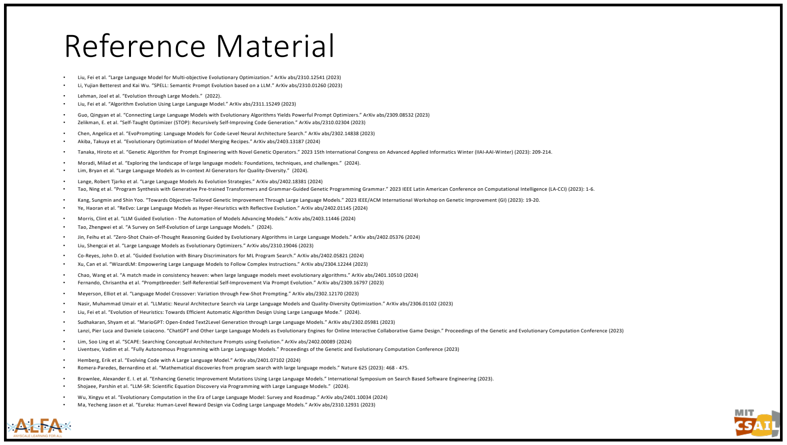
Algorithm Variants:

- How can we probe LLM-EA to understand the limits of its literal coding competence and more pragmatic coding competences?
- How can an LLM-EA algorithm integrate design explorations related to cooperation, modularity, reuse, or competition?
- How can an LLM-EA algorithm model biology differently from EAs?
- How can an LLM-EA intrinsically, or with guidance, support open-ended evolution?
- What new variants hybridizing EA, LLM-EA and/or another search heuristic are possible and in what respects are they advantageous?
- Is there a relevant multi-objective optimization and many-objective optimization approach with LLM-EA?

Analysis Avenues:

- How well does LLM-EA scale with population size and problem complexity?
- What is a search space in LLM-EA and how can it be characterized with respect to problem difficulty?
- To what extent does an LLM-based approach intrinsically address novelty or quality-diversity?
- What is the most accurate computational complexity of LLM-EA?

42



Reference Material

- Liu, Fei et al. "Large Language Model for Multi-objective Evolutionary Optimization." ArXiv abs/2310.12541 (2023)
- Li, Yujian Betterest and Kai Wu. "SPELL: Semantic Prompt Evolution based on a LLM." ArXiv abs/2310.01260 (2023)
- Lehman, Joel et al. "Evolution through Large Models." (2022).
- Liu, Fei et al. "Algorithm Evolution Using Large Language Model." ArXiv abs/2311.15249 (2023)
- Guo, Qingyan et al. "Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers." ArXiv abs/2309.08532 (2023)
- Zelikman, E. et al. "Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation." ArXiv abs/2310.02304 (2023)
- Chen, Angelica et al. "EvoPrompting: Language Models for Code-Level Neural Architecture Search." ArXiv abs/2302.14838 (2023)
- Akiba, Takuya et al. "Evolutionary Optimization of Model Merging Recipes." ArXiv abs/2403.13187 (2024)
- Tanaka, Hiroto et al. "Genetic Algorithm for Prompt Engineering with Novel Genetic Operators." 2023 15th International Congress on Advanced Applied Informatics Winter (IIAI-AAI-Winter) (2023): 209-214.
- Moradi, Milad et al. "Exploring the landscape of large language models: Foundations, techniques, and challenges." (2024).
- Lim, Bryan et al. "Large Language Models as In-context AI Generators for Quality-Diversity." (2024).
- Lange, Robert Tjarko et al. "Large Language Models As Evolution Strategies." ArXiv abs/2402.18381 (2024)
- Tao, Ning et al. "Program Synthesis with Generative Pre-trained Transformers and Grammar-Guided Genetic Programming Grammar." 2023 IEEE Latin American Conference on Computational Intelligence (LA-CCI) (2023): 1-6.
- Kang, Sungmin and Shin Yoo. "Towards Objective-Tailored Genetic Improvement Through Large Language Models." 2023 IEEE/ACM International Workshop on Genetic Improvement (GI) (2023): 19-20.
- Ye, Haoran et al. "ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution." ArXiv abs/2402.01145 (2024)
- Morris, Clint et al. "LLM Guided Evolution - The Automation of Models Advancing Models." ArXiv abs/2403.11446 (2024)
- Tao, Zhengwei et al. "A Survey on Self-Evolution of Large Language Models." (2024).
- Jin, Feihu et al. "Zero-Shot Chain-of-Thought Reasoning Guided by Evolutionary Algorithms in Large Language Models." ArXiv abs/2402.05376 (2024)
- Liu, Shengcai et al. "Large Language Models as Evolutionary Optimizers." ArXiv abs/2310.19046 (2023)
- Co-Reyes, John D. et al. "Guided Evolution with Binary Discriminators for ML Program Search." ArXiv abs/2402.05821 (2024)
- Xu, Can et al. "WizardLM: Empowering Large Language Models to Follow Complex Instructions." ArXiv abs/2304.12244 (2023)
- Chao, Wang et al. "A match made in consistency heaven: when large language models meet evolutionary algorithms." ArXiv abs/2401.10510 (2024)
- Fernando, Chrisantha et al. "Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution." ArXiv abs/2309.16797 (2023)
- Meyerson, Elliot et al. "Language Model Crossover: Variation through Few-Shot Prompting." ArXiv abs/2302.12170 (2023)
- Nasir, Muhammad Umair et al. "LLMatic: Neural Architecture Search via Large Language Models and Quality-Diversity Optimization." ArXiv abs/2306.01102 (2023)
- Liu, Fei et al. "Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Mode." (2024).
- Sudhakaran, Shyam et al. "MarioGPT: Open-Ended Text2Level Generation through Large Language Models." ArXiv abs/2302.05981 (2023)
- Lanzi, Pier Luca and Daniele Loiacono. "ChatGPT and Other Large Language Models as Evolutionary Engines for Online Interactive Collaborative Game Design." Proceedings of the Genetic and Evolutionary Computation Conference (2023)
- Lim, Soo Ling et al. "SCAPE: Searching Conceptual Architecture Prompts using Evolution." ArXiv abs/2402.00089 (2024)
- Liventsev, Vadim et al. "Fully Autonomous Programming with Large Language Models." Proceedings of the Genetic and Evolutionary Computation Conference (2023)
- Hemberg, Erik et al. "Evolving Code with A Large Language Model." ArXiv abs/2401.07102 (2024)
- Romera-Paredes, Bernardino et al. "Mathematical discoveries from program search with large language models." Nature 625 (2023): 468 - 475.
- Brownlee, Alexander E. I. et al. "Enhancing Genetic Improvement Mutations Using Large Language Models." International Symposium on Search Based Software Engineering (2023).
- Shojaee, Parshin et al. "LLM-SR: Scientific Equation Discovery via Programming with Large Language Models." (2024).
- Wu, Xingyu et al. "Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap." ArXiv abs/2401.10034 (2024)
- Ma, Yecheng Jason et al. "Eureka: Human-Level Reward Design via Coding Large Language Models." ArXiv abs/2310.12931 (2023)

43