Domain Centered FMs

Team 2

Computer Science
The University of Virginia

April 11, 2024

Jessie Chen (hc4vb)

Agenda

1. **Software Engineering**
2. **Bioengineering**
3. **Law**

Language Models for Software Engineering: A Systematic Literature Review

Presenters: Jessie (hc4vb) and Soneya (sh7hv)

UNIVERSITY of VIRGINIA

Overview

1. **Software Engineering**
   (a) SE is a discipline focused on the development, implementation, and maintenance of software systems.
   (b) The utilization of LLMs in SE emerges from the perspective where numerous SE challenges can be effectively reframed into data, code, or text analysis tasks.
2. **Contributions**
   (a) covers 229 papers published between 2017 and 2023.
   (b) summarizes usage and trends of different LLM categories within the SE domain.
   (c) describes the data processing stages.
   (d) discusses optimizers used.
   (e) analyzes key applications of LLMs in SE
   (f) presents key challenges and opportunities

What LLMs have been employed?

There are more than 50 different LLMs used for SE tasks in the papers collected.

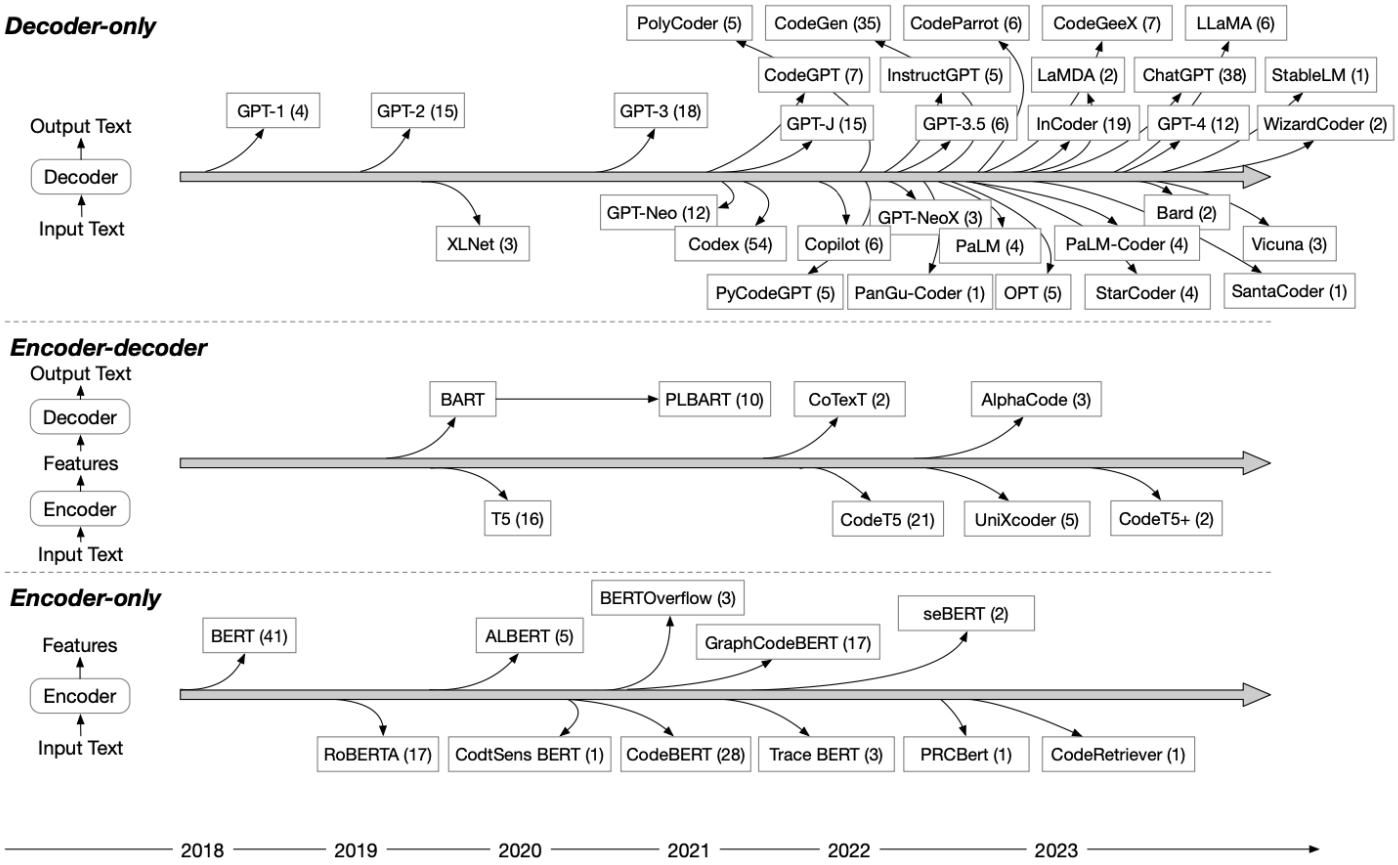


Figure: Distribution of the LLMs discussed in the collected papers. The numbers in parentheses indicate the count of papers in which each LLM has been utilized.

What LLMs have been employed?

1. **Evolution of LLM architectures in 2021:** We see the emergence of decoder-only and encoder-decoder models in 2021.
2. **Diversity of LLM architectures in 2022:** 2022 experienced a significant increase in diversity, with more varied LLM architectures finding representation.
3. **Dominance of the decoder-only architecture in 2023:** 2023 signaled a strong shift towards decoder-only LLMs.

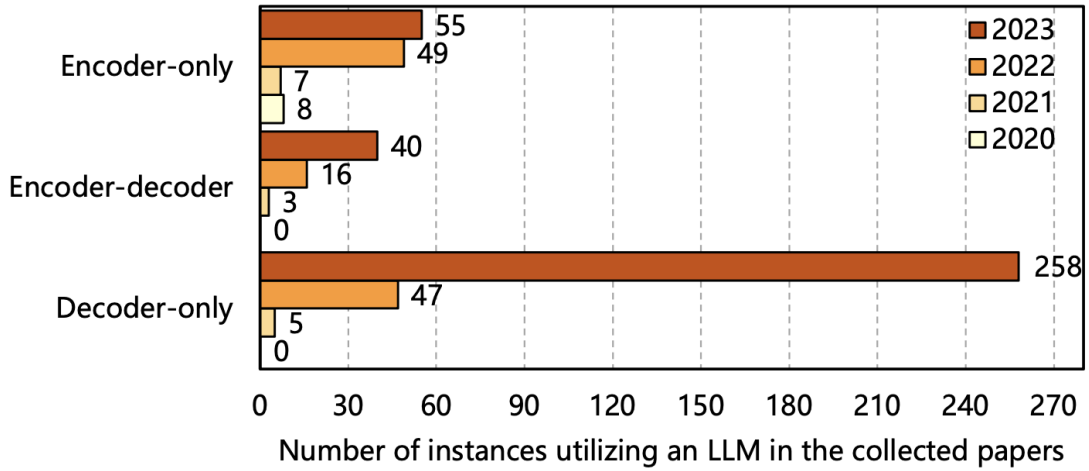


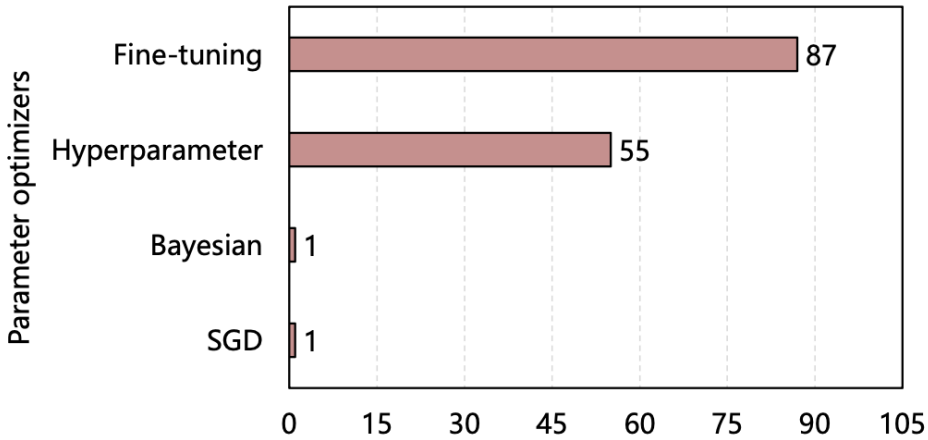
Figure: Trends in the application of LLMs with different architectures in SE tasks over time

What types of SE datasets have been used in existing LLM4SE studies?

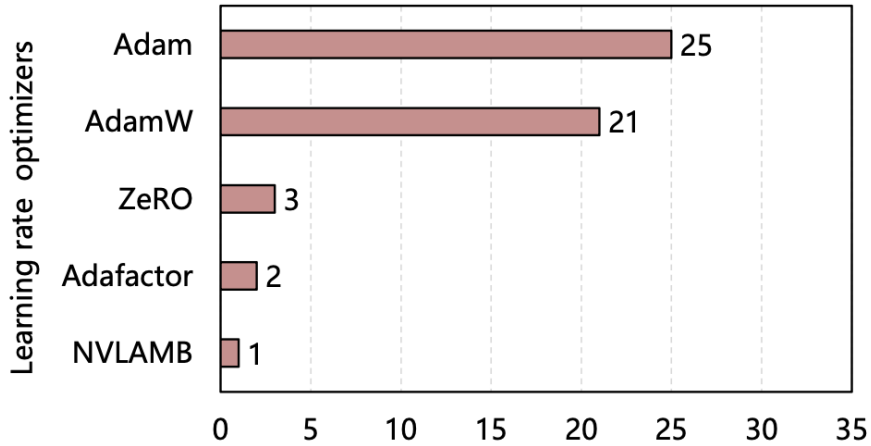
Category	Data type	# Studies	Total	References
Code-based datasets	Source code	44	70	[4] [16] [27] [29] [32] [35] [46] [81] [82] [96] [112] [118] [124] [127] [135] [156] [167] [180] [185] [201] [211] [221] [229] [235] [245] [257] [275] [287] [288] [290] [293] [303] [324] [337] [344] [356] [348] [361] [371] [379] [387] [393] [399] [403]
	Bugs	4		[136] [144] [352] [354]
	Patches	4		[162] [309] [310] [390]
	Code changes	4		[84] [171] [249] [383]
	Test suites/cases	3		[70] [126] [389]
	Error code	3		[117] [250] [347]
	Vulnerable source code	2		[30] [42]
	Bug-fix pairs	2		[67] [372]
	Labeled clone pairs	1		[61]
	Buggy programs	1		[28]
	Packages	1		[283]
	Flaky test cases	1		[72]
Text-based datasets	Prompts	28	104	[29] [58] [60] [62] [85] [128] [129] [139] [140] [165] [172] [200] [282] [292] [294] [302] [304] [305] [312] [316] [320] [329] [331] [332] [345] [358] [367] [394]
	Programming problems	14		[38] [33] [107] [132] [155] [168] [176] [277] [279] [295] [311] [335] [377] [375]
	SO (i.e., Stack Overflow) posts	9		[22] [102] [103] [105] [115] [152] [263] [343] [392]
	Bug reports	9		[45] [73] [88] [105] [122] [137] [163] [197] [220]
	Programming tasks (and solutions)	7		[54] [59] [70] [143] [153] [231] [278]
	Requirements documentation	6		[66] [109] [151] [199] [217] [338]
	APIs/API documentation	5		[50] [141] [248] [359] [376]
	Q&A pairs	5		[259] [284] [321] [329] [402]
	Vulnerability descriptions	4		[10] [251] [308] [351]
	Bug reports with changesets	4		[44] [50] [105] [131]
	Methods	3		[209] [211] [374]
	Code comments	2		[257] [357]
	Project issues	2		[78] [298]
	Software specifications	1		[205]
	Dockerfiles	1		[108]
	Semantic merge conflicts	1		[382]
	Site text	1		[149]
	User intents	1		[125]
	User reviews	1		[339]
Graph-based datasets	GUI Images	1	1	[151]
Software repository-based datasets	Issues and commits	3	5	[7] [183] [392]
	Pull-requests	2		[285] [392]
Combined datasets	Source code and comments	8	18	[91] [158] [234] [257] [276] [318] [384] [388]
	Programming tasks and test suites/cases	4		[192] [236] [243] [291]
	Binary code and related annotations	1		[6]
	Failing test code and error messages	1		[355]
	Source code and Q&A pairs	1		[280]
	Source code and test suites/cases	1		[317]
	Source code, methods, and logging statements	1		[178]
	Source code, description, and code environment	1		[184]

What techniques are used to optimize LLM4SE?

1. Fine-tuning emerges as the most widely used parameter optimization algorithm.
2. Adam is the most frequently used learning rate optimizer.
3. Prompt engineering is the most advantageous in providing task-relevant knowledge and enhancing LLMs' versatility and efficacy.



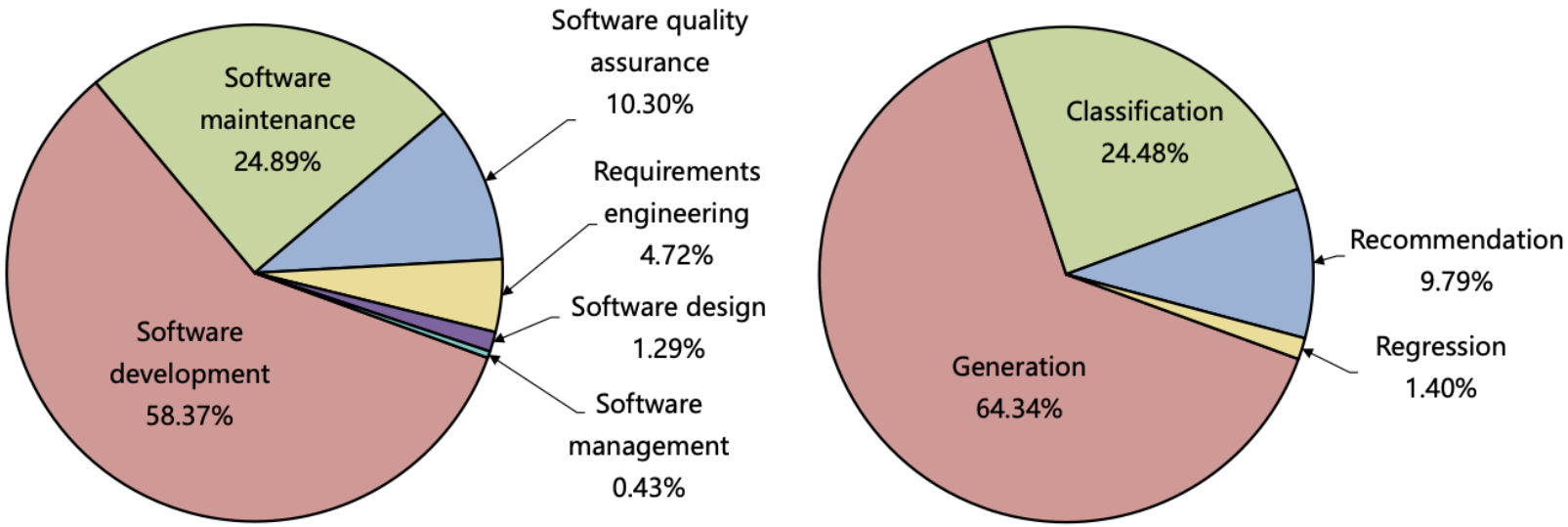
(a) Parameter optimizers used in collected studies.



(b) Learning rate optimizers used in prior studies.

What SE tasks have been efficiently addressed by LLMs?

1. The tasks are grouped based on the six phases of the Software Development Life Cycle (SDLC).
2. The highest number of studies is observed in software development.
3. The majority of studies, about 64.34%, center around generation tasks.



(a) Distribution of LLM usages in SE activities. (b) Problem classification based on collected studies.

Soneya Binta Hossain (sh7hv)

UNIVERSITY of VIRGINIA

Distribution of SE tasks over six SE activities.

Table 11. Distribution of SE tasks over six SE activities.

SE Activity	SE Task		Total
Requirements engineering	Anaphoric ambiguity treatment (3)	Requirements term identification (1)	11
	Requirements classification (3)	Coreference detection (1)	
	Requirement analysis and evaluation (2)	Traceability automation (1)	
Software design	GUI retrieval (1)	Software specification synthesis (1)	3
	Rapid prototyping (1)		
Software development	Code generation (62)	Agile story point estimation (1)	136
	Code completion (16)	API documentation smell detection (1)	
	Code summarization (10)	API entity and relation extraction (1)	
	Code understanding (7)	Code optimization (1)	
	Code search (5)	Code example recommendation (1)	
	Program synthesis (5)	Control flow graph generation (1)	
	API recommendation (2)	Data analysis (1)	
	API synthesis (2)	Identifier normalization (1)	
	Code comment generation (2)	Instruction generation (1)	
	Code representation (2)	Type inference (1)	
	Method name generation (2)	Others (11)	
Software quality assurance	Test generation (8)	Bug localization (1)	24
	Vulnerability detection (7)	Failure-inducing test identification (1)	
	Test automation (4)	Flaky test prediction (1)	
	Verification (2)		
Software maintenance	Program repair (23)	Duplicate bug report detection (1)	58
	Code review (6)	Decompilation (1)	
	Debugging (4)	Program merge conflicts repair (1)	
	Bug report analysis (3)	Sentiment analysis (1)	
	Code clone detection (3)	Tag recommendation (1)	
	Logging (2)	Vulnerability repair (1)	
	Bug prediction (1)	Commit classification (1)	
	Bug triage (1)	Traceability recovery (1)	
	Bug report replay (1)	Others (6)	
Software management	Effort estimation (1)		1

SE Activity 1: Requirements Engineering

Anaphoric Ambiguity: Varying interpretations by readers of the same natural language requirement.

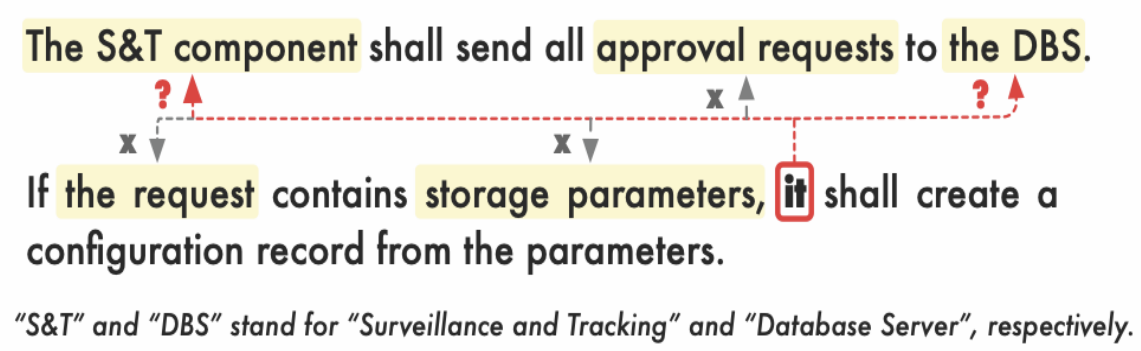


Figure 1: Example of Anaphoric Ambiguity.

SE Activity 1: Requirements Engineering

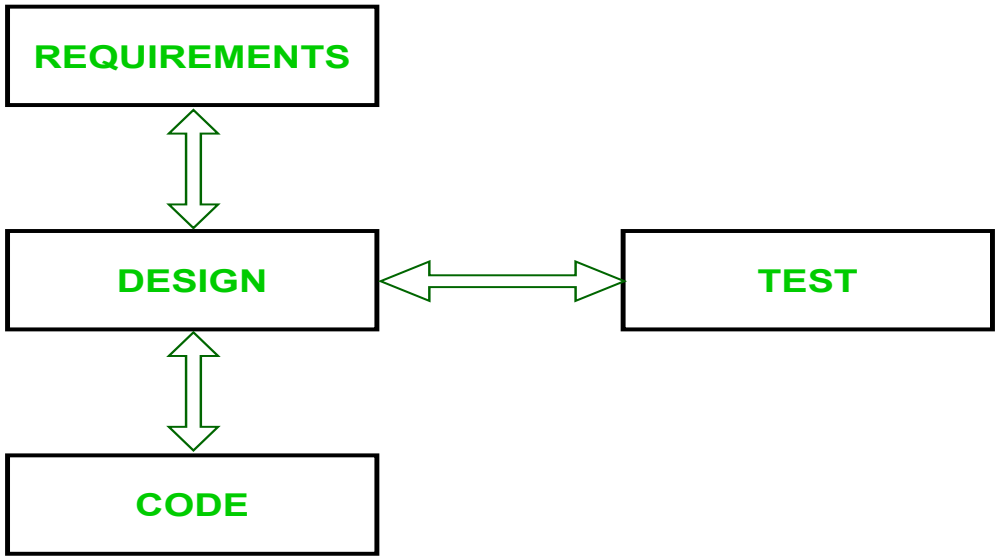
Anaphoric ambiguity treatment:

- Can cause misinterpretations, affecting later development stages.
- LLMs like BERT and SpanBERT can efficiently resolve such ambiguity [Moharil et al. and Ezzini et al.]
- ChatGPT showed remarkable ability in detecting and resolving this issue [Sridhara et al.]

SE Activity 1: Requirements Engineering

Traceability Automation

- Traceability links SE artifacts: requirements, designs, code, and tests

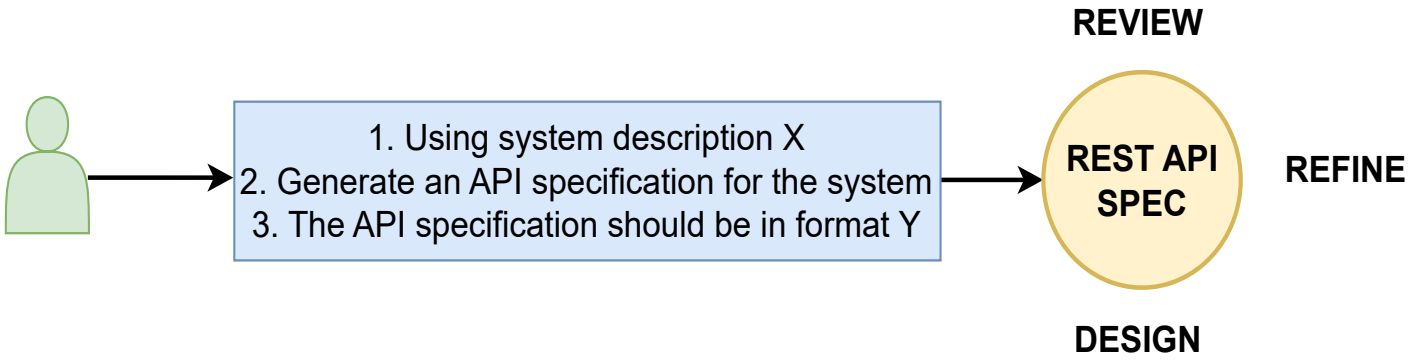


- T-BERT can generate trace links between source code and natural language artifacts with few training instances.

SE Activity 2: Software Design

Rapid Prototyping

- Enables developers to quickly visualize and iterate on software designs, accelerating the development process.



- White et al. showed that LLMs can be leveraged for rapid prototyping by introducing prompt design patterns.

SE Activity 2: Software Design

Software Specification Synthesis

- Software specification is a valuable artifact, but manually synthesizing specification is very costly.
- LLM can automatically synthesize specification from natural language, outperforming previous SOTA by 21% in F1 score.

NL Spec: Users can upload photos to their profile, but only JPG and PNG files are allowed. Each photo must be less than 5MB in size.

Formal Spec:
$\forall Photo(upload(Photo) \rightarrow (fileType(Photo, JPG) \vee fileType(Photo, PNG)))$
$\forall Photo\,(upload(Photo) \rightarrow fileSize(Photo, Size) < 5)$

SE Activity 3: Software Development

Wide range of SE tasks where LLMs have been applied for far.

Software development	Code generation (62)	Agile story point estimation (1)	136
	Code completion (16)	API documentation smell detection (1)	
	Code summarization (10)	API entity and relation extraction (1)	
	Code understanding (7)	Code optimization (1)	
	Code search (5)	Code example recommendation (1)	
	Program synthesis (5)	Control flow graph generation (1)	
	API recommendation (2)	Data analysis (1)	
	API synthesis (2)	Identifier normalization (1)	
	Code comment generation (2)	Instruction generation (1)	
	Code representation (2)	Type inference (1)	
	Method name generation (2)	Others (11)	

SE Activity 3: Software Development

Code Generation with LLMs

- Code generation has long been a task of interest.

Create a function that takes a list of numbers as input and returns the sum of all the even numbers in the list.

```
def sum_of_even_numbers(numbers):
    """
    Calculate the sum of all even numbers in a list.

    Parameters:
    numbers (list of int): A list of integers.

    Returns:
    int: The sum of all even numbers in the list.
    """
    sum_even = 0
    for num in numbers:
        if num % 2 == 0:
            sum_even += num
    return sum_even
```

SE Activity 3: Software Development

Code Generation with LLMs

Table 12. The state-of-the-art applications of LLMs in code generation task.

Model	Baseline	Benchmark	Metric	Date	Reference
GPT-3.5	Codex, CodeGen, CodeGeeX, LLaMA, InCoder, PyCodeGPT, CodeParrot, GPT-2	HumanEval, MBPP, MBCPP	Pass@k	May 11, 2023	[169]
GPT-4	PaLM Coder, Codex, CodeGen-Mono, Incoder, CodeGeeX, AlphaCode	HumanEval, HumanEval-ET, MBPP, MBPP-ET	Pass@k	May 24, 2023	[61]
GPT-4	GPT-3.5, StarCoder, CodeGen, CodeGen2, Vicuna, SantaCoder, Incoder, GPT-J, GPT-Neo, PolyCoder, StableLM	HumanEval, HumanEval+, HumanEval-mini	Pass@k	Jun 12, 2023	[187]
GPT-4	GPT-3.5, WizardCoder, Instruct-StarCoder, SantaCoder, Instruct-CodeGen, CodeGeeX, InCoder, Vicuna, ChatGLM, PolyCoder	ClassEval, HumanEval	Pass@k	Aug 3, 2023	[63]

- LLMs are effective in method-level generation, with ongoing research to improve class-level generation accuracy.
- The integration of LLMs with SE tools and practices presents new opportunities for collaborative software development.

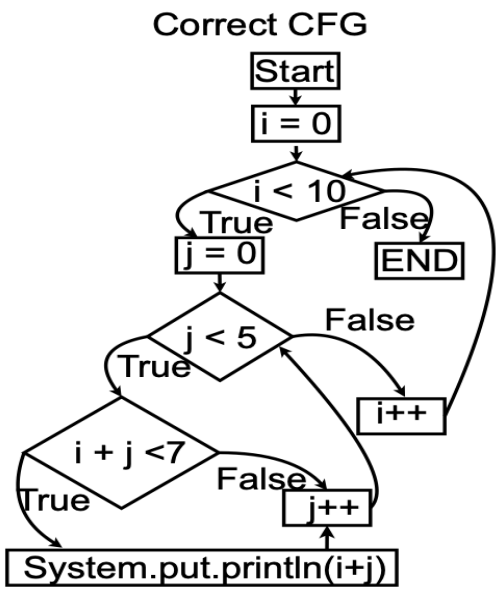
SE Activity 3: Software Development

Control Flow Graph Generation with LLMs

- Control Flow Graphs (CFGs) are sequences of statements and their execution order.
- Critical in many SE tasks: code search, clone detection, code classification.

Code

```
for (int i=0; i ? 10; i++){
  for (int j=0;j<5;j++){
    if(i+j<7);
      System.out.println(i+j);
    }
  }
```



SE Activity 3: Software Development

Control Flow Graph Generation with LLMs

- Huang et al. introduced a novel LLM-based approach for generating behaviorally correct CFGs from partial code, using Chain of Thoughts (CoT).

- CoT works in four steps: structure hierarchy, nested block extraction, individual CFG generation, and CFG fusion.

- LLM-based method achieves superior node and edge coverage in CFGs, demonstrating the potential of LLMs in enhancing program analysis techniques.

SE Activity 4: Software Quality Assurance

Test generation

- Automates test case creation.

Example Test Cases:
assertEqual(sum_of_even_numbers([1, 2, 3, 4, 5, 6]), 12)
assertEqual(sum_of_even_numbers([1, 3, 5, 7]), 0)

- LLMs generate diverse test cases, achieve good coverage, detect unique bugs.

- NLD to test generation improves collaboration between developers and testers.

- LLMs identify test coverage gaps and suggest relevant test cases to close them.

SE Activity 4: Software Quality Assurance

Failure-Inducing Test Identification.

- Distinguishing between pass-through and fault-inducing test cases is crucial for debugging.

```
def merge_sorted_lists(list1, list2):

    merged_list = []
    i, j = 0, 0
    while i < len(list1) and j < len(list2):
        if list1[i] < list2[j]:
            merged_list.append(list1[i])
            i += 1
        else:
            merged_list.append(list2[j])
            j += 1
    merged_list.extend(list1[i:])
    # Bug: Forgets to merge remaining elements from list2
    return merged_list
```

```
TEST 1: Cannot Find the Bug

list1 = [1, 3, 5]
list2 = [2, 4]
expected = [1, 2, 3, 4, 5]
assert merge_sorted_lists(list1, list2) == expected,


TEST 2:Failure Inducing

list1 = [1, 2]
list2 = [3, 4, 5]
expected = [1, 2, 3, 4, 5]
assert merge_sorted_lists(list1, list2) == expected
```

- ChatGPT can effectively detect subtle code discrepancies and generate fault-inducing test cases.

SE Activity 5: Software Maintenance

Program Repair with LLMs.

- LLMs can be leveraged for automated bug identification and fixing.
- BERT, CodeBERT, Codex and GPT series excel in generating correct patches.
- Incorporating additional context can boost LLM's program repair performance.

```
def sum_of_even_numbers(numbers):
    sum_even = 0
    for num in numbers:
        if num % 2:
            sum_even += num
    return sum_even
```

```
def sum_of_even_numbers(numbers):
    sum_even = 0
    for num in numbers:
        if num % 2==0:
            sum_even += num
    return sum_even
```

SE Activity 6: Software Management

Effort estimation.

1. Effort estimation is crucial for planning the time, resources, and manpower needed for software projects.

2. BERT's showed potential to significantly aid in the accurate prediction of resources and manpower needed for software maintenance, streamlining project planning and resource allocation.

Summary

- SE tasks categorized into six areas show LLMs' diverse applications.
- LLM usage spans 55 SE tasks, predominantly in software development, with minimal application in software management.
- Code generation and program repair emerge as prominent LLM tasks.

Challenges

- Model size and deployment
- Data dependency (lack of data, overlapping issue, privacy issue)
- Ambiguity in code generation
- Generalizability
- Evaluation metrics issue (typical metrics)
- Interpretability, Trustworthiness, and Ethical Usage

Exploring the Impact of Large Language Models (LLMs) on Bioengineering

Presenter: Minjae Kwon (hbt9su)

The Genetic Dogma

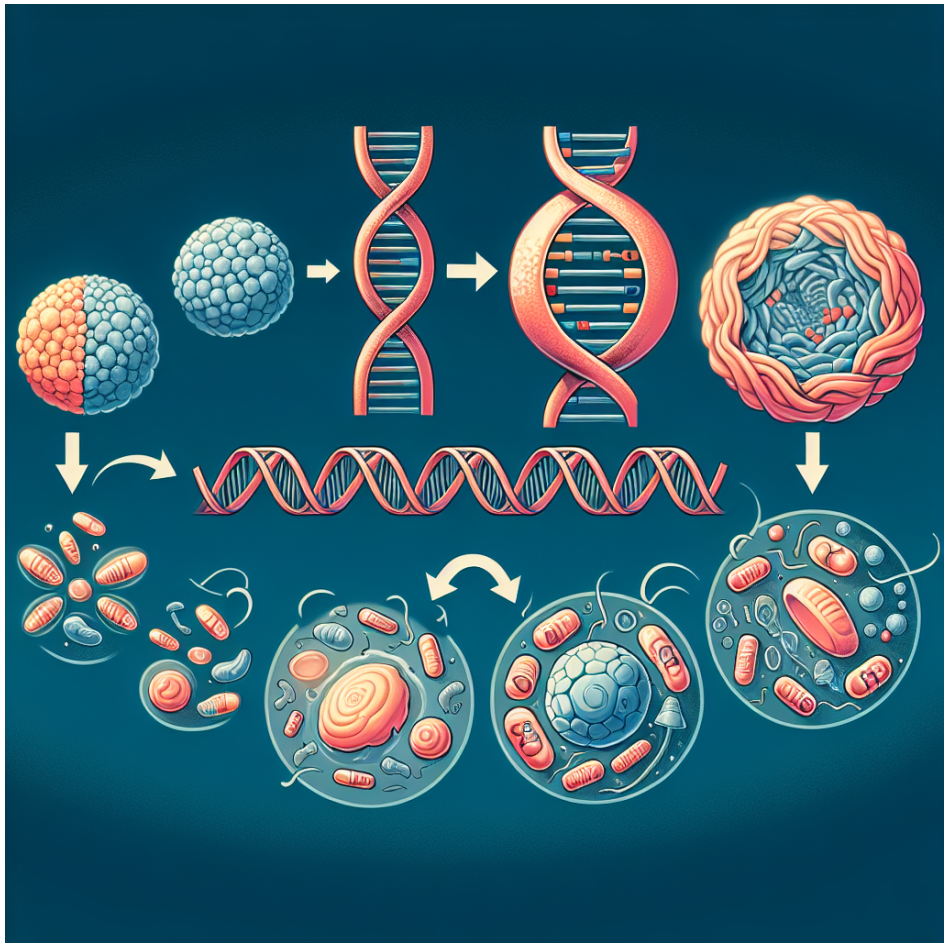


Figure: The biological trajectory of any organism (e.x. Human) is a complex interplay between genetics and environment[1]

[1] Image from DALL·E 3 with prompt: Gradually changing image from Nucleotide → DNA → Gene → Chromosome → Cell → Organism

Basic Terms

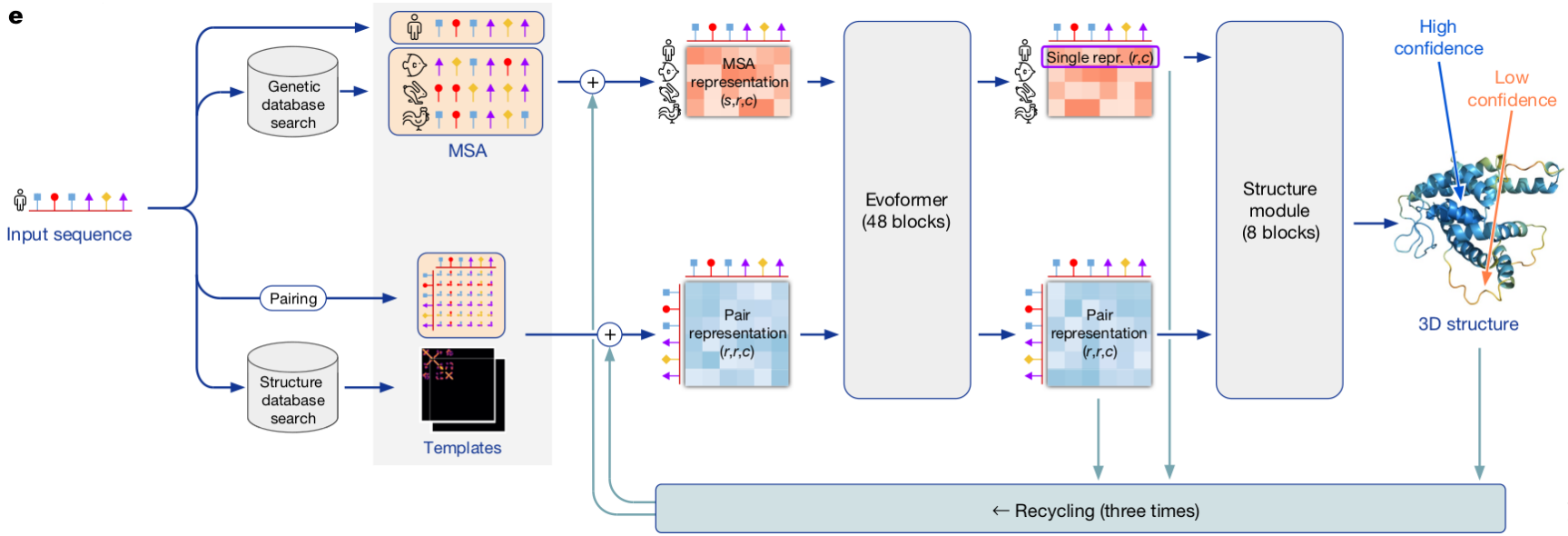
Terms

1. **Nucleic Acids**: Macromolecules for the storage, transmission, and expression of genetic information.
2. **Nucleotide**: Building blocks of nucleic acids such as DNA (deoxyribonucleic acid) and RNA (ribonucleic acid).
3. **Amino Acids**: Building blocks of proteins for biological processes such as the synthesis of proteins, enzymes, hormones, and neurotransmitters.
4. **Residues**: A specific unit or component within a larger molecule, such as a protein or nucleic acid
5. **Genetic Code**: A set of rules that defines the correspondence between the nucleotide sequence of a DNA or RNA molecule and the amino acid sequence of a protein.

AlphaFold

Principles about the Folding of Protein Chains (Novel Prize Lecture)

A protein's amino acid sequence should fully determine its structure.

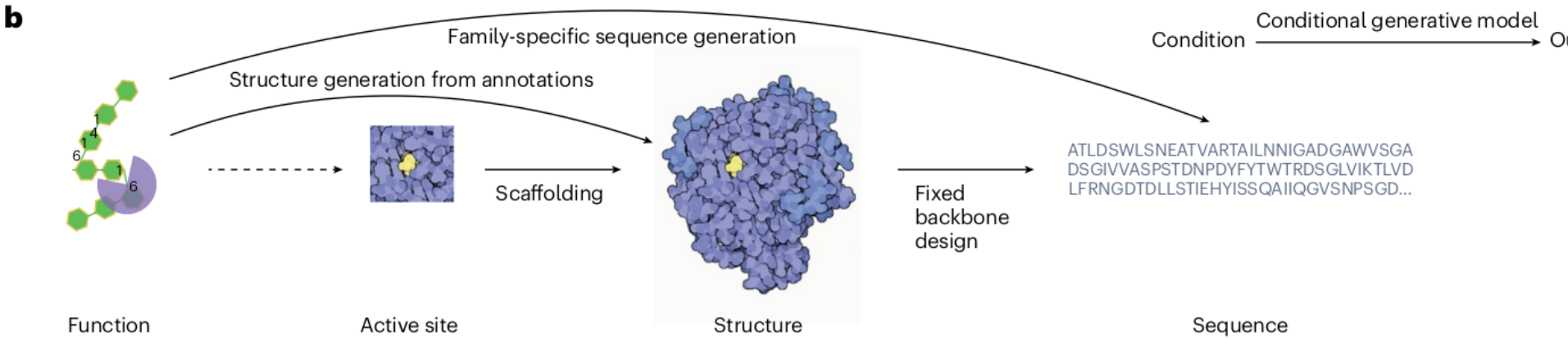


[2]

[2] Highly accurate protein structure prediction with AlphaFold

Pre-Training Process[4]

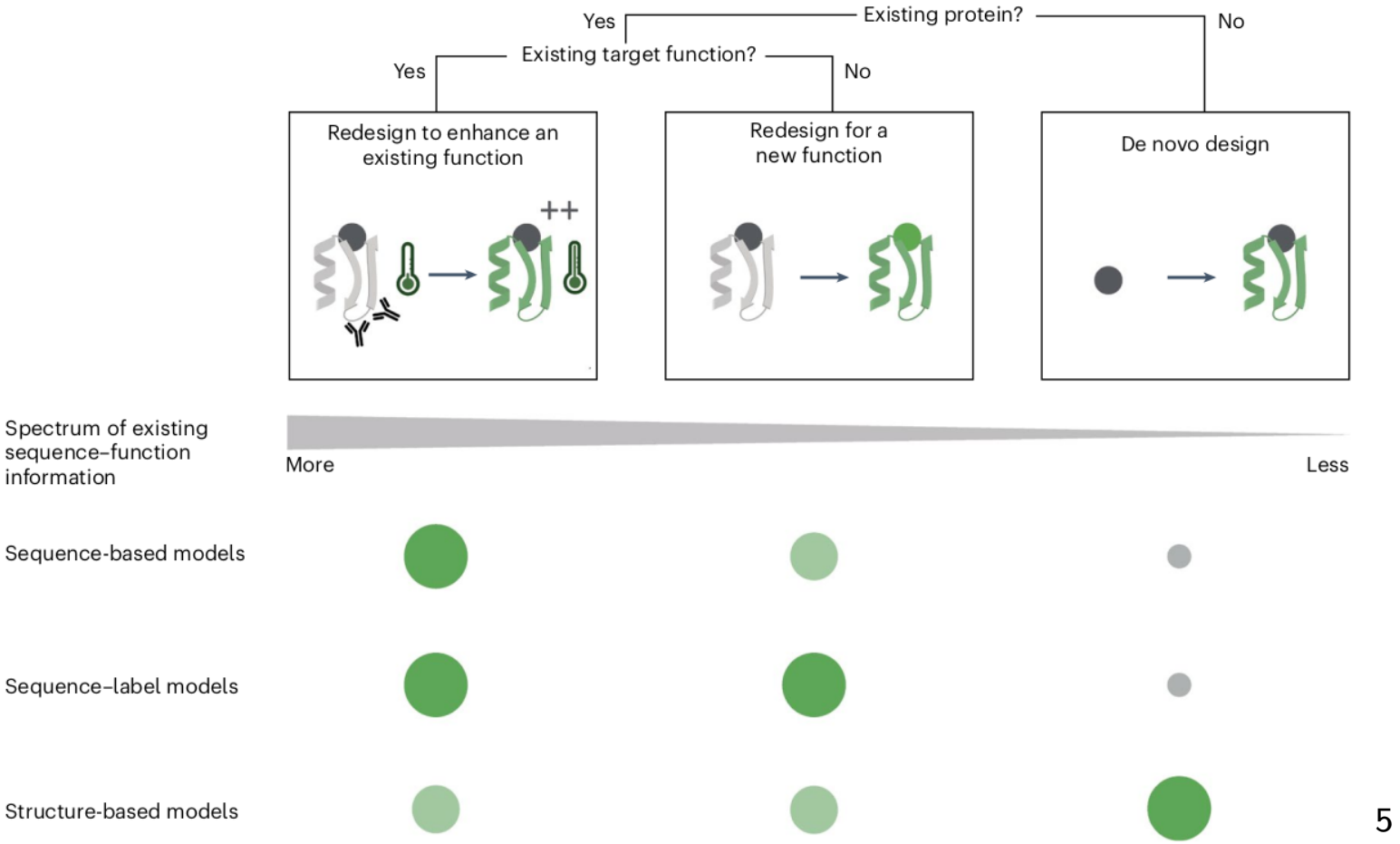
1. **Sequence of Amino Acids**: $a = (a_1, \ldots, a_{n_a})$.
2. **Control Tags**: $c = (c_1, \ldots, c_{n_c})$.
3. **Input**: $x = [c; a]$ concatenation of a and c.
4. **Objective Function**: $L(D) = -\frac{1}{|D|} \sum_{k=1}^{|D|} \frac{1}{n^k} \sum_{i=1}^{n^k} \log p_\theta \left(x_i^k \mid x_{<i}^k \right)$.
5. **Generation**: $p_\theta \left(a_1 \mid \underline{c} \right), p_\theta \left(a_2 \mid \underline{a_1}, \underline{c} \right), \ldots, p_\theta \left(a_j \mid \underline{a_{<j}}, \underline{c} \right)$ conditioned on a control tag sequence $\underline{c}$.



[3] Generative models for protein structures and sequences

[4] Large language models generate functional protein sequences across diverse families

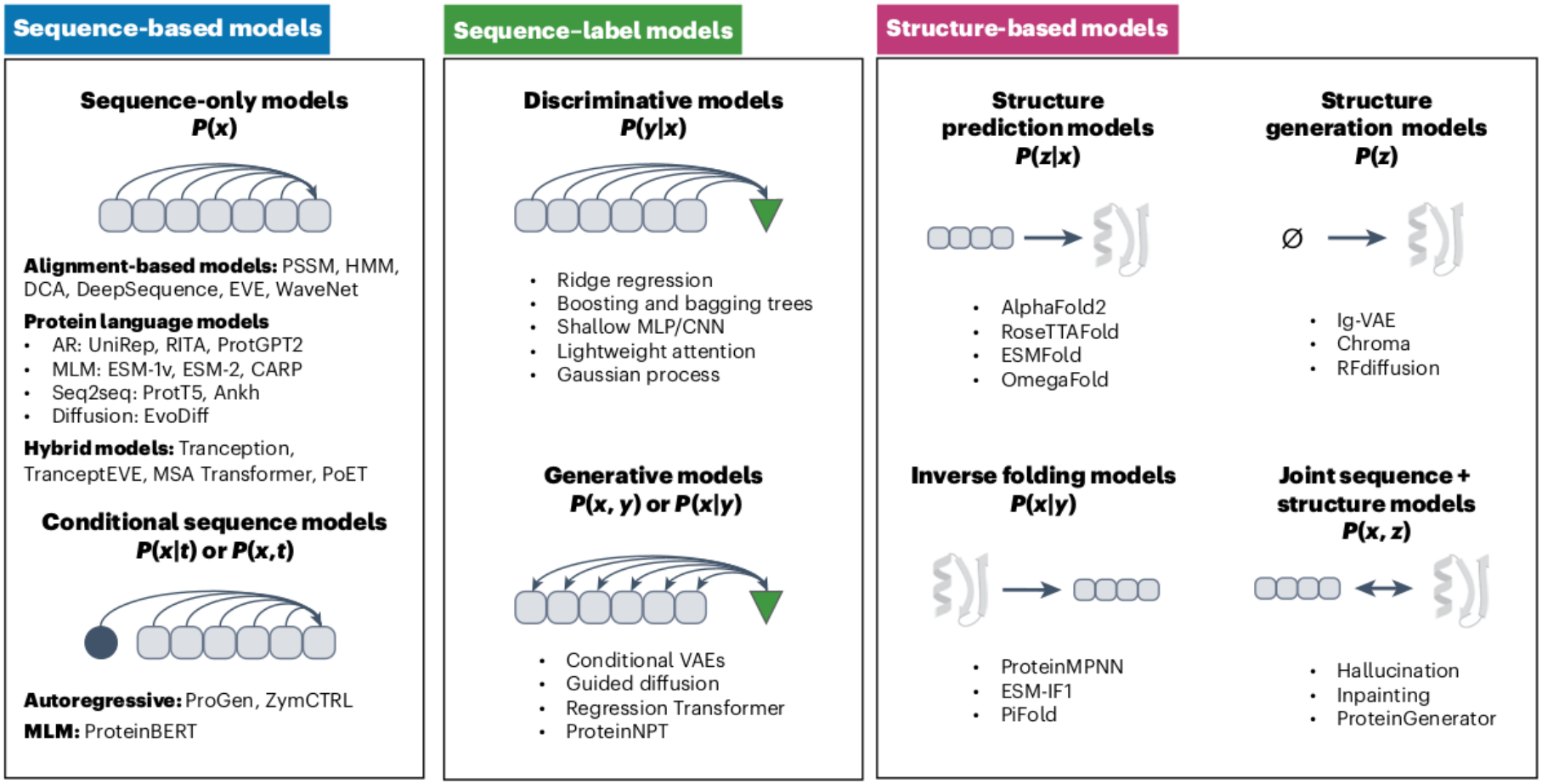
Protein Objective



[5]

[5] Machine learning for functional protein design

Topology of Protein Design



6

[6] Machine learning for functional protein design

A Genomic Foundation Model

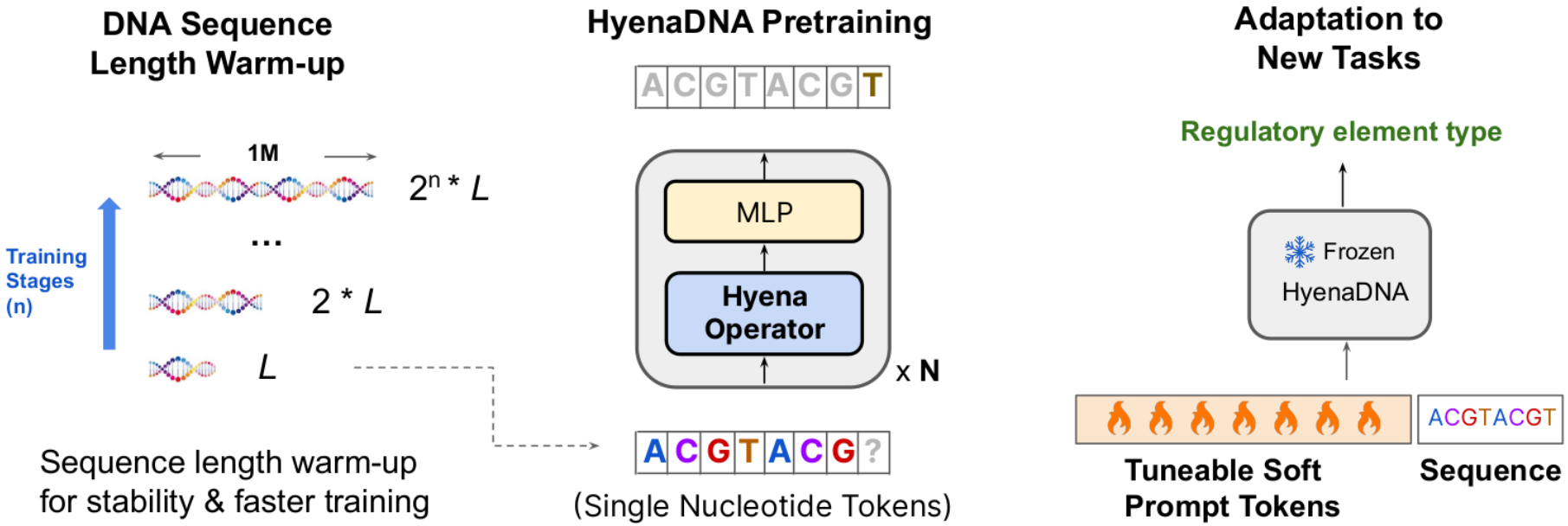
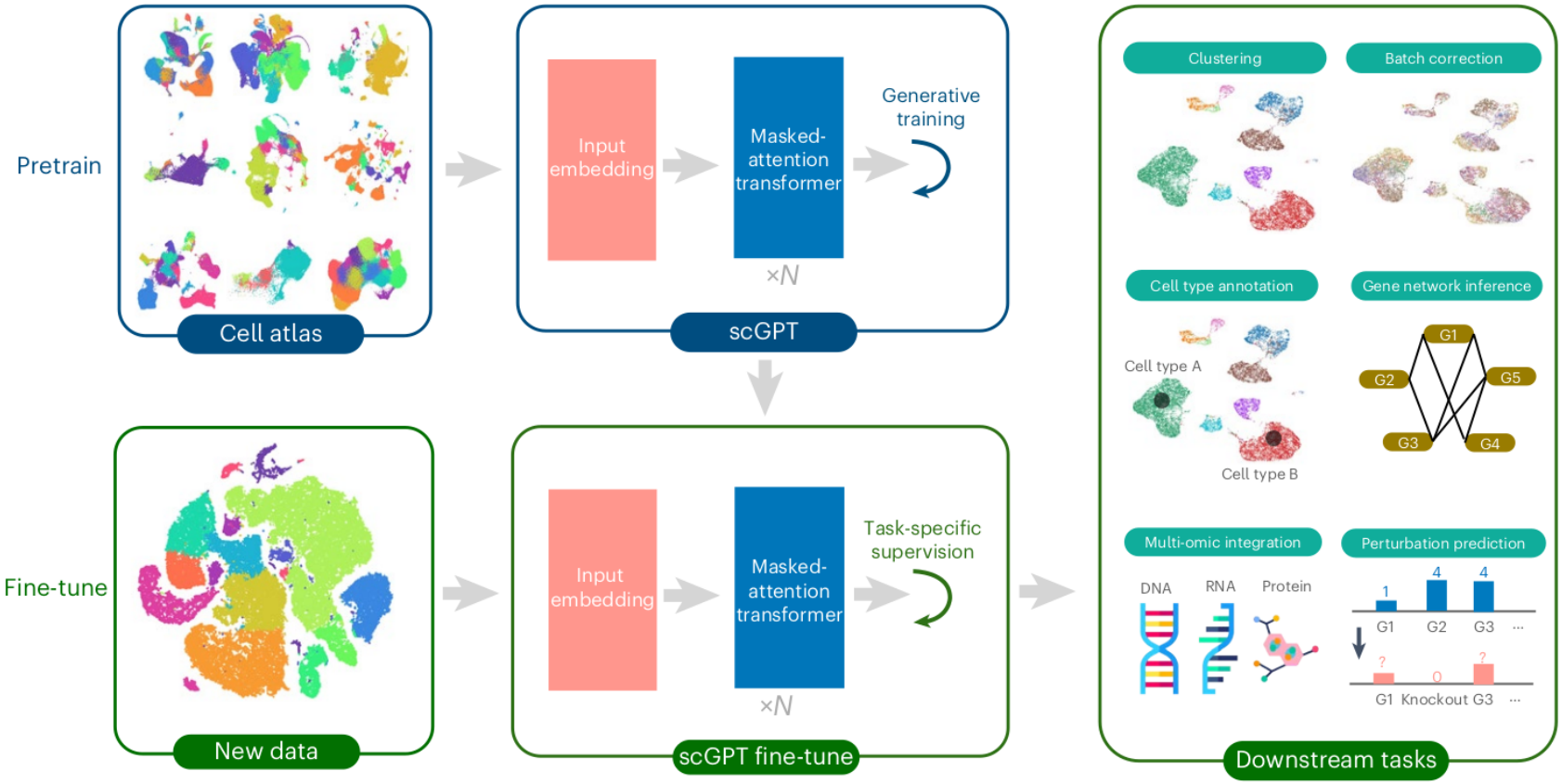


Figure: CNN-combined Model.[7] This beats previous Transformer-based Model[8]

[7] HyenaDNA: Long-Range Genomic Sequence Modeling at Single Nucleotide Resolution

[8] The Nucleotide Transformer: Building and Evaluating Robust Foundation Models for Human Genomics

A Single-Cell Foundation Model

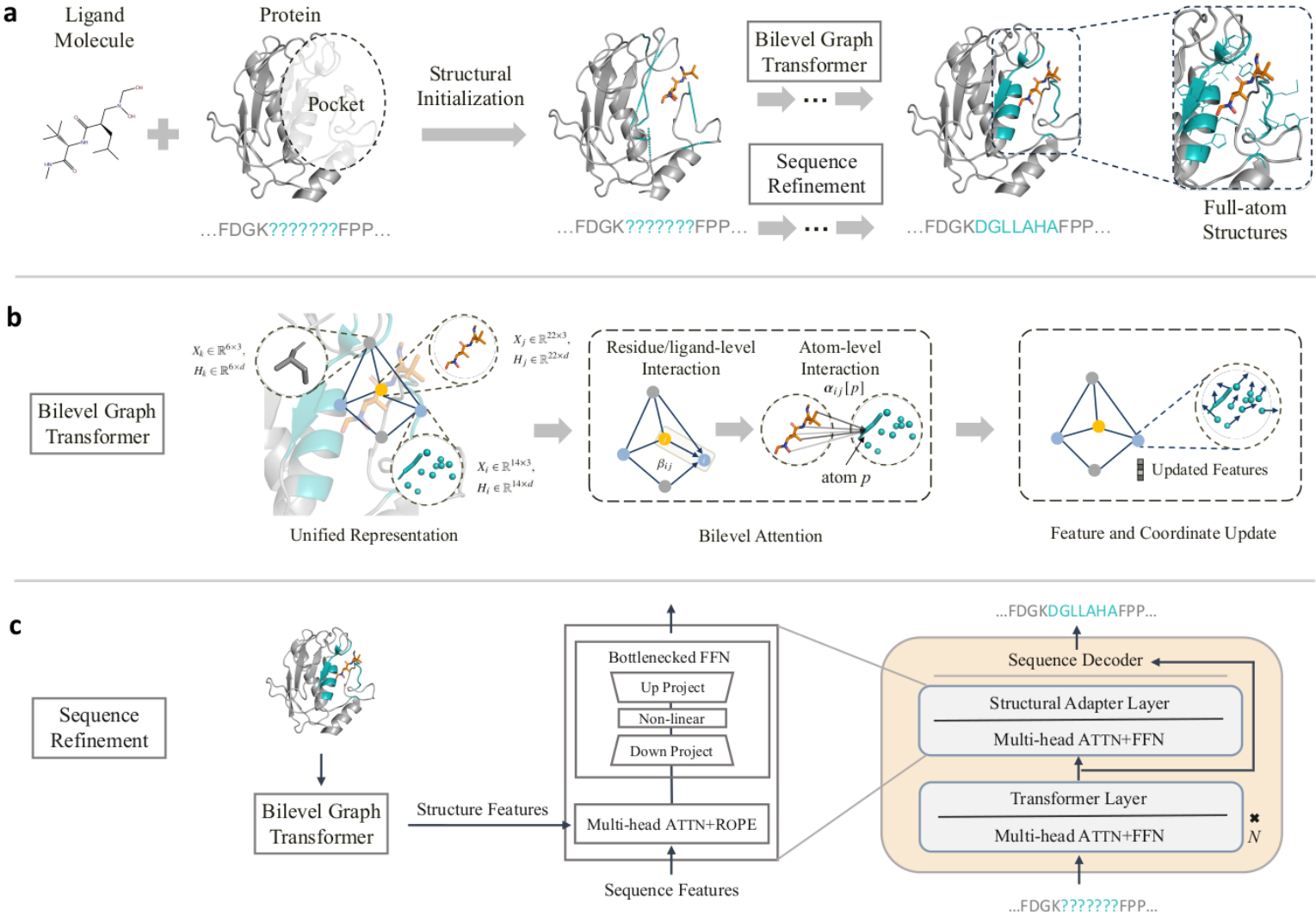


Figure

9

[9] scGPT: toward building a foundation model for single-cell multi-omics using generative AI

Design of Full-atom Ligand-binding Protein Pockets



[10] PocketGen: Generating Full-Atom Ligand-Binding Protein Pockets

Protein Structure Generation

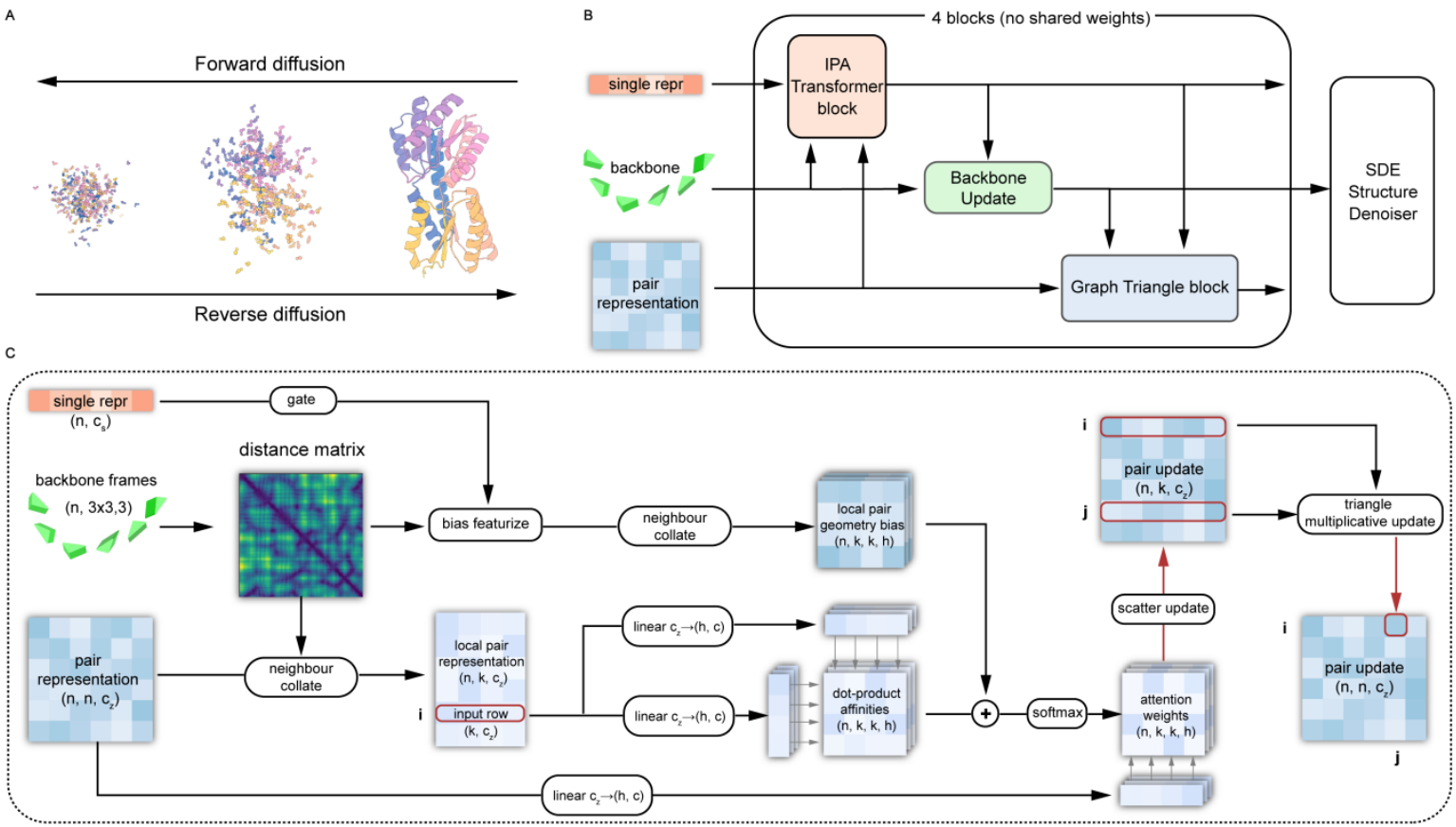
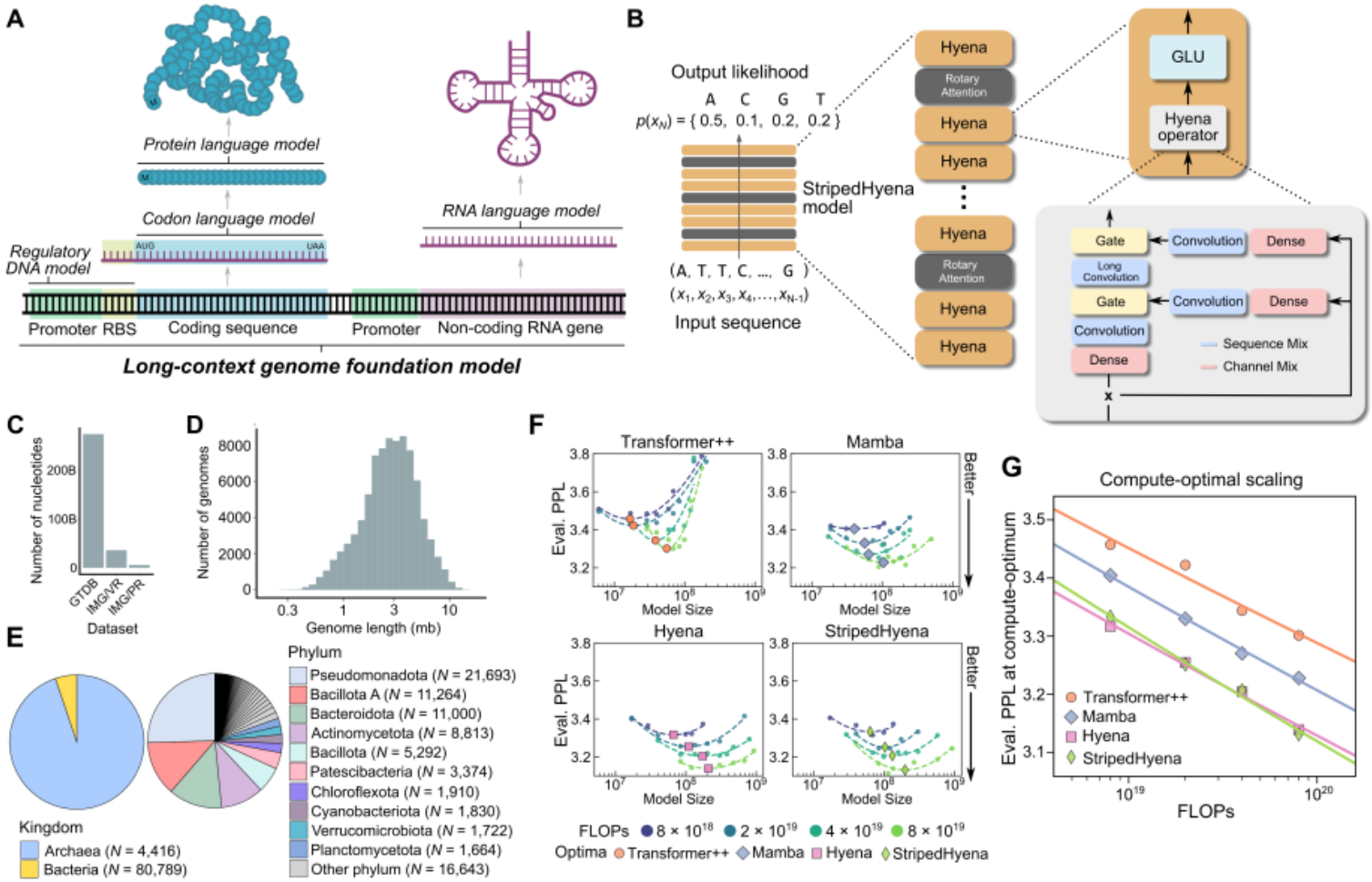


Figure: Diffusion-based Model.

11

[11] Proteus: exploring protein structure generation for enhanced designability and efficiency

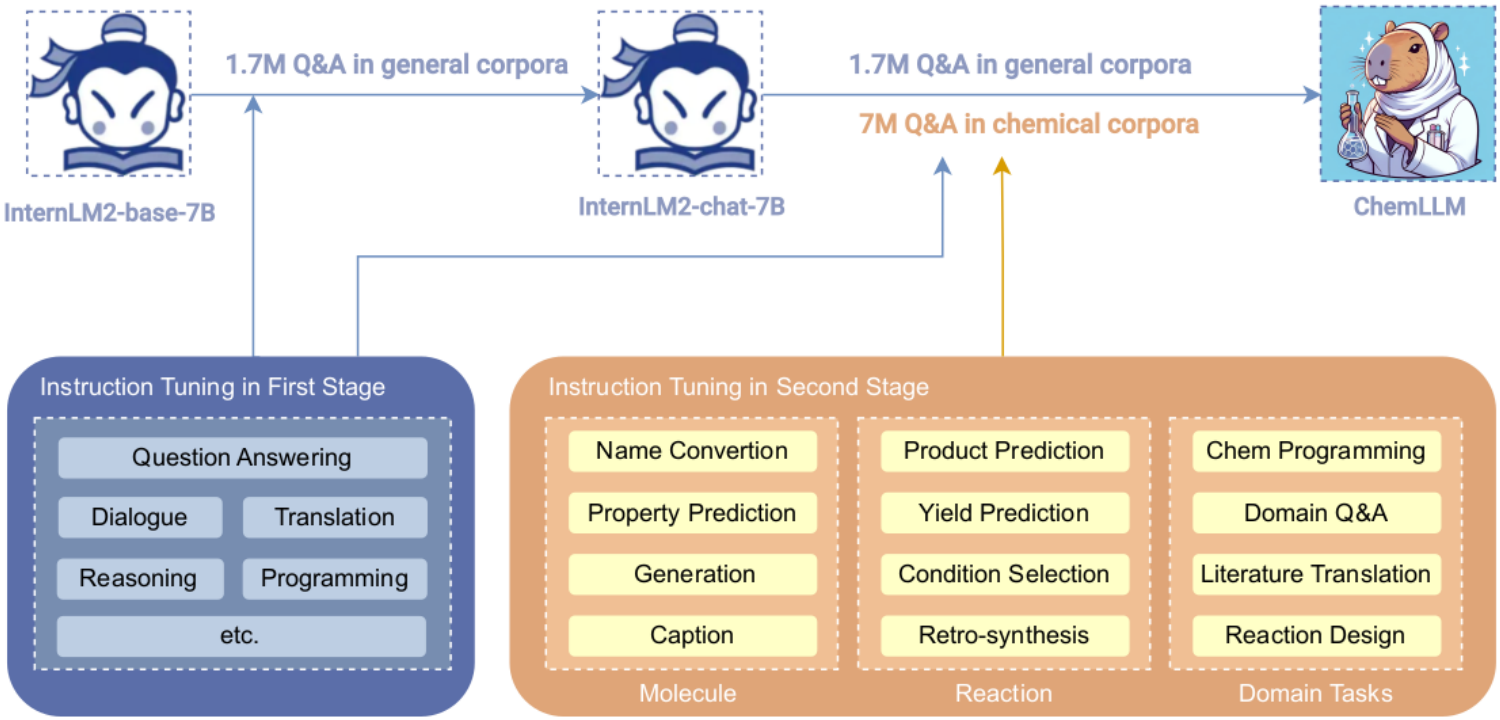
Molecular to Genome



12

[12] modeling and design from molecular to genome scale with Evo

ChemLLM



13

[13] ChemLLM: A Chemical Large Language Model

Large Language Models in Law: A Survey

Presenters: Jeffrey Chen (fyy2ws), Ali Zafar Sadiq (mzw2cu)

Jeffrey Chen (fyy2ws)

Overview

1. **Contributions**
2. **Survey Overview**
3. **Evolution of Judicial Technology**
4. **Recent Applications**
5. **Challenges**
6. **Future Directions**
7. **Conclusions**

1. Contributions

1. **The first comprehensive review article on legal LLMs**
2. **Demonstrate use of legal LLMs**
3. **Provide the latest research on legal LLMs**
4. **Summarize the key challenges and future directions of legal LLMs**

2. Survey Overview

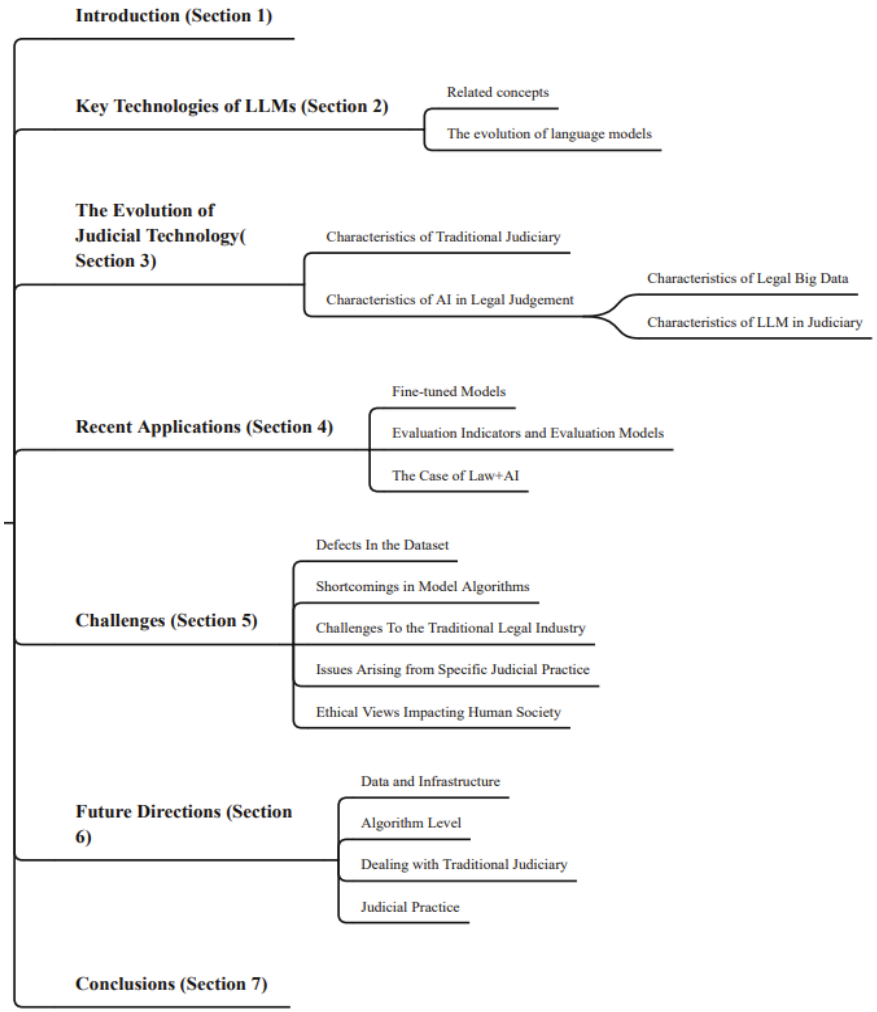


Figure 1: The outline of our overview.

3.1 Characteristics of Traditional Judicial System

* Reliance on human decision-making
* Precedent-based
* Flexibility
* Time and resource-consuming

3.2 Characteristics of AI in Legal Judgement

1. **Characteristics of Legal Big Data**
   * Unstructured
   * Multilingual and multicultural
   * Vast scale and Complexity
   * Timeliness
   * Data multi-sourcing
   * Privacy and security concerns

3.2 Characteristics of AI in Legal Judgement (cont.)

2. Characteristics of LLMs in Judiciary

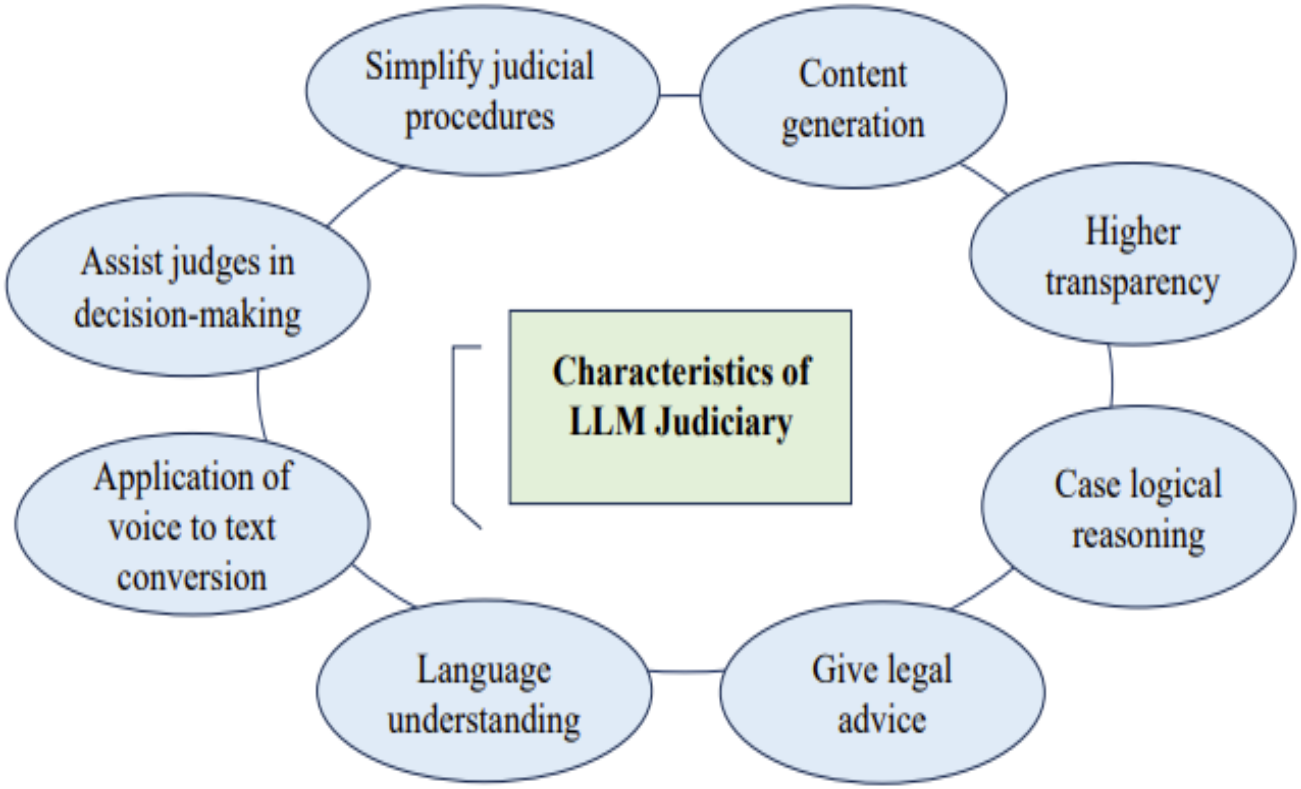


Figure 3: Characteristics of LLM in Judiciary.

4.Recent Applications

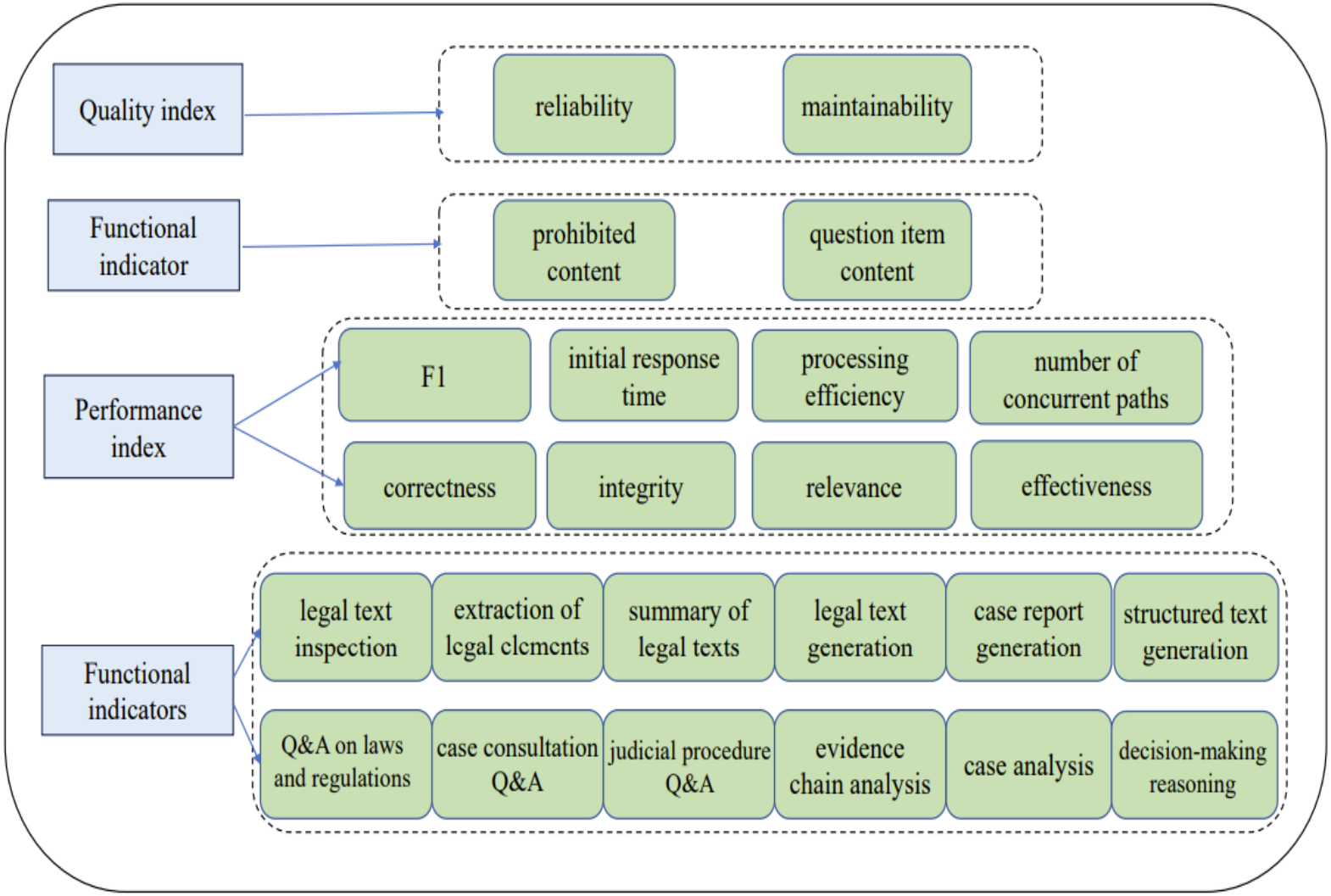
1. Fine-tuned Models

LLM-based law	Traditional model	Website
LawGPT_{zh}	ChatGLM-6B LoRA 16bit	https://github.com/LiuHC0428/LAW-GPT
LaWGPT	Chinese-LLaMA, ChatGLM	https://github.com/pengxiao-song/LaWGPT
LexiLaw	ChatGLM-6B	https://github.com/CSHaitao/LexiLaw
Lawyer LLaMA	LLaMA	https://github.com/AndrewZhe/lawyer-llama
HanFei	BLOOMZ-7B1	https://github.com/siat-nlp/HanFei
ChatLaw	Jiangzi-13B, Anima-33B	https://github.com/PKU-YuanGroup/ChatLaw
Lychee	GLM-10B	https://github.com/davidpig/lychee_law
JurisLMs	AI Judge, AI Lawyer	https://github.com/seudl/JurisLMs
Fuzi.mingcha	ChatGLM	https://github.com/irlab-sdu/fuzi.mingcha

Figure: Ten Popular Fine-Tuned Legal LLMs

4. Recent Applications (cont.)

2. Evaluation Indicators for the AI and law system



5.1 Defects in Datasets

(1) Inadequate Data Aquisition
  * Insufficient sources of judicial data and documents
  * Insufficient sharing of legal data
  * Non-standard legal documents

(2) Inaccurate Interpretation of Legal Concepts

(3) Dataset Characteristics
  * Timeliness
  * Credibility
  * Scalability

5.2 Shortcomings in Algorithms

(1) Interpretability

* Insufficient interpretability reduces people's trust in the judicial application of AI

(2) Ethics, bias, and fairness

* Algorithms may contain elements of inequality
* Insufficient security in algorithm outsourcing
* Reduced transparency of LLMs in law may lead to judicial unfairness
* Algorithmic Bias

Ali Zafar Sadiq (mzw2cu)

5.3 Challenges of Traditional Legal Industry

1. **Neglecting Judicial Independence**

(a) In terms of legal enforcement: it includes

* Interpreting Civil Law
* Explaining uncertain concepts, and evaluating disputes

(b) In terms of fact-finding: use of discretion, subjective judgment, experiential

Legal LLMs lead to

(a) Overly relying on AI

(b) Form preconceived notions

5.3 Challenges of Traditional Legal Industry

For Example:

In **assessing the compensation amount** in civil litigation, judges can comprehensively consider factors such as the extent of the **victim's financial loss** and the **defendant's ability to compensate**.

In contrast, the algorithms of legal LLMs struggle to measure the extent of loss

Legal LLMs can **assist judges**
It **does not possess professional judicial experience** and cannot independently make judgments in cases.

5.3 Challenges of Traditional Legal Industry

2. Impact on Judicial System

Legal LLMs have restrained the subjective initiative of judges and the development of traditional trial systems as reflected in:

(1) Court idleness:

* Restrict the **subjective initiative of judges**
* Diminish the **solemnity of the legal process**

(2) Crisis in the hierarchy of trial: Legal AI systems will impact the judicial process in the hierarchical system.

For example,
Any party **dissatisfied with any judgment of a lower court** can **appeal to a higher court** which with legal AI system remains same.

5.4 Issues Arising from Specific Judicial Practice

1. The lack of universality in applications

Legal LLMs often extract feature values from cases and search for similar cases within existing multidimensional datasets to find the "optimal solution"

Legal regulations may vary across different countries or regions, leading to inconsistent decision outcomes for the same case under different legal rules, so, the "optimal solution" proposed by the large model may not apply to a particular case.

2. The lack of subjective thinking, emotions, and experience

Legal LLMs lack autonomous thinking abilities and professional experience, among other things.

Judicial decision making process is not merely a logical reasoning process on a single layer but also involves moral, ethical, and practical considerations in the legal system.

5.4 Issues Arising from Specific Judicial Practice

3. **Contradiction with the presumption of innocence principle**
   * COMPAS system for **crime prediction and risk assessment**
   * PredPol for iterative **calculation of potential crime locations**
   * PRECOBS system in Germany for **burglary prevention and violence crime prediction**

Figure: *Futuristic System that apprehends people based on their probability of committing Crime.*

5.4 Issues Arising from Specific Judicial Practice

3. Contradiction with the presumption of innocence principle

* Imbalance of prosecution and defense
* Unequal control over data
* Differences in the ability to analyze case data
* Issues of policy attention, investment imbalance, and unequal exploration
* Administrative Performance

5.4 Issues Arising from Specific Judicial Practice

4. **Privacy infringement**

Legal LLMs may collect excessive user data, which can lead to privacy breaches when the input information is sensitive.

5. **Issues of intellectual property identification and protection.**

It becomes unclear whether credit should be given to the human users who utilize AI or to the AI system itself.

5.5. Ethical Views Impacting Human Society

1. **Disregard for human subjectivity:**

Human subjectivity is susceptible to algorithmic bullying.

2. **Misleading user comments:**

In testing certain LLMs, such as ChatGPT, AI has displayed behaviors such as inducing users to divorce, making inappropriate comments, and even encouraging users to disclose personal privacy or engage in illegal activities

3. **Ethical value consistency:**

There may be situations where AI misleads or harms human interests.

6. Future Directions

(1) Data and Infrastructure

* Obtaining more comprehensive legal big data
* Defining the boundaries of legal concepts and limiting the scope of application
* Data transparency
* Building a legal knowledge graph
* Optimizing the foundational infrastructure for model training [High-performance computing resources, Storage and data management, Model scaling and deployment etc.]

(2) Algorithm Level:

* Strategy adjustment and optimized algorithm
* Limiting algorithmic biases and "black box" operations the scope of application
* Promote limited algorithmic transparency

UNIVERSITY of VIRGINIA

6. Future Directions

(3) Dealing with Traditional Judiciary

* Clarifying the positioning of large models
* Defining the thinking capability of LLMs
* Ensuring parties' access to data
* Expanding and optimizing the consulting function of judicial large models

(4) Judicial Practice:

* Improve accountability mechanisms to prevent political interference
* Foster the development of interdisciplinary talents
* Collaboration and sharing of experiences

7. Conclusions

This paper synthesized various technologies and ideas regarding the opportunities, challenges, and recommendations for the application of AI in the judicial field.

REFERENCES

https://arxiv.org/abs/2308.10620
https://arxiv.org/abs/2312.03718
https://arxiv.org/abs/2306.15794
https://arxiv.org/abs/2402.06852
https://www.nature.com/articles/s41586-021-03819-2
https://www.nature.com/articles/s41587-023-02115-w
https://www.nature.com/articles/s41587-022-01618-2
https://www.nature.com/articles/s41587-024-02127-0
https://www.biorxiv.org/content/10.1101/2024.02.10.579791v2
https://www.biorxiv.org/content/10.1101/2024.02.25.581968v1
https://www.biorxiv.org/content/10.1101/2023.04.30.538439v1
https://www.biorxiv.org/content/10.1101/2023.01.11.523679v1
https://www.biorxiv.org/content/10.1101/2024.02.27.582234v1

UNIVERSITY of VIRGINIA