npj | computational materials

Brief communication

Published in partnership with the Shanghai Institute of Ceramics of the Chinese Academy of Sciences

https://doi.org/10.1038/s41524-024-01449-6

# Large language models design sequence-defined macromolecules via evolutionary optimization



Wesley F. Reinhart [1,2] ✉ & Antonia Statt[3] ✉

We demonstrate the ability of a large language model to perform evolutionary optimization for materials discovery. Anthropic's Claude 3.5 model outperforms an active learning scheme with handcrafted surrogate models and an evolutionary algorithm in selecting monomer sequences to produce targeted morphologies in macromolecular self-assembly. Utilizing pre-trained language models can potentially reduce the need for hyperparameter tuning while offering new capabilities such as self-reflection. The model performs this task effectively with or without context about the task itself, but domain-specific context sometimes results in faster convergence to good solutions. Furthermore, when this context is withheld, the model infers an approximate notion of the task (e.g., calling it a protein folding problem). This work provides evidence of Claude 3.5's ability to act as an evolutionary optimizer, a recently discovered emergent behavior of large language models, and demonstrates a practical use case in the study and design of soft materials.

Self-assembly of macromolecules plays an important role in technology (e.g., nano-patterning[1]) and biology (e.g., protein folding[2]). Macromolecules with precisely controlled sequences of repeat units are a promising new material class due to their incredible tunability[3]. While organisms use them to encode genetic material via DNA[2], recent advances in chemistry enabled synthetic analogs[3]. However, their tunability also presents a challenge; the number of possible nucleotide sequences in a short DNA strand is astronomical and cannot be evaluated by brute force.

We previously[4,5] studied the self-assembly of model copolymers with hydrophobic $A$ and hydrophobic $B$ units, exhibiting rich self-assembly behavior due to complex interactions between many nearby chains[4,6]. We quantified these structures using unsupervised learning and predicted their self-assembly using recurrent neural network (RNNs)[5]. However, this required a training set of ~$10^3$ examples, which, while relatively small for machine learning, is large for computational science.

With continuing improvements in Large Language Models (LLMs), research activity around emergent reasoning capability within these models increased. Specifically, recent studies investigated the ability of LLMs to perform evolutionary optimization[7–10]. For small-scale optimization problems such as the Traveling Salesman Problem, LLMs can compete with handcrafted heuristic algorithms[8], a surprising result considering these are not native natural language problems. As black-box optimizers, LLMs matched the performance of multi-objective evolutionary optimization[9,10]. Many interesting observations regarding the performance of LLMs on these tasks can be found in refs. 7–10, among others. This newly discovered phenomenon should receive more attention in the near future, hopefully including some explanation for how and why it occurs.

We implement an LLM-driven evolutionary optimizer to design morphologies of macromolecules from the sequence, like the inverse protein folding problem. Crucially, the LLM is entirely pre-trained, without any fine-tuning on domain-specific data; we simply use Anthropic's commercially available model, which excels on a wide variety of tasks[11]. We utilize our unsupervised representation [4] (Fig. 1a) to produce an order parameter $Z$ from sequence vector $x$ via MD. An approximation of this mapping, $f(x) \rightarrow \tilde{Z}$, was learned by a RNN[5] (Fig. 1c). We take this approximation as the ground truth and search for optimal sequences $x\prime$ to minimize the distance to target $\tilde{Z}_t$, meaning designing a sequence that self-assembles into a precisely defined structure. We do not validate the end-to-end $f(x) \rightarrow Z$ (i.e., MD output) here, focusing on the identification of optimal $\tilde{Z}$.

This LLM-based design is illustrated in Fig. 1d. This requires no prior examples, performing the optimization entirely on the fly. We compare the performance of the LLM (Claude 3.5 Sonnet) to two other algorithms: active learning (using an ensemble of Random Forest regressors) and evolutionary algorithm (using two-point crossover). A robust body of literature explores both active learning[12] and evolutionary algorithms[13] in materials science applications, such as autonomous laboratories[14]. These strategies utilize

[1]Department of Materials Science and Engineering, Pennsylvania State University, University Park, 16802 PA, USA. [2]Institute for Computational and Data Sciences, Pennsylvania State University, University Park, 16802 PA, USA. [3]Department of Materials Science and Engineering, Grainger College of Engineering, University of Illinois Urbana-Champaign, Champaign, 61801 IL, USA. ✉e-mail: reinhart@psu.edu; statt@illinois.edu

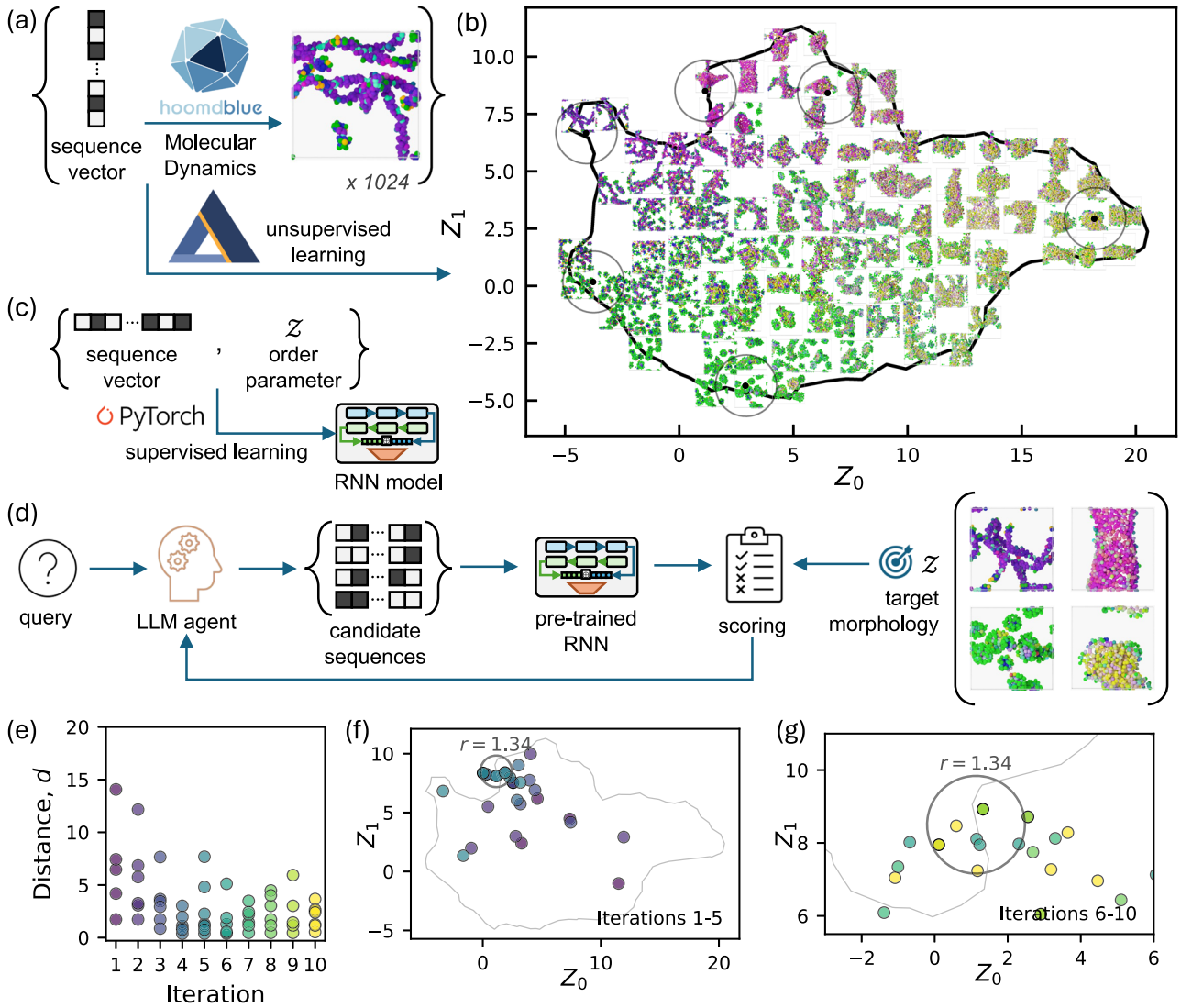


**Fig. 1 | Schematic of LLM-driven evolutionary optimization. a** The sequence-defined macromolecular aggregation problem from our prior work[4]. A monomer sequence is injected into an MD simulation, which yields a snapshot. The snapshots are evaluated with unsupervised learning to produce a 2D order parameter, *Z*. **b** An illustrated version of the *Z* latent space, with simulation snapshots centered on their corresponding embedding. The mean distance between adjacent snapshots is 1.34 units, the same as the threshold in the sequence selection task below. The target morphologies used in the task are shown as points with a radius of 1.34 units drawn around them. **c** The supervised machine learning model from our prior work[5], wherein the monomer sequence is mapped to *Z* by an RNN. **d** Based on a query, an LLM agent proposes a batch of monomer sequences, which are evaluated by the pre-trained RNN model. Each candidate's score, i.e., the distance to the target morphology, is returned to the LLM to create an iterative optimization loop. **e** A representative rollout from the LLM evolutionary optimization using the "scientific" prompt. Points represent labeled samples at each iteration. Colors are redundant with the *x* axis in this panel and act as a legend for the following panels. **f** The same rollout from **e** showing the position of labeled sequences as a function of LLM iteration in the first five iterations. The target is indicated in the same fashion as (**b**). Colors correspond to iterations in (**e**). **g** The sample rollout from (**f**) over the last five iterations, with the latent space zoomed in on the target region. Colors correspond to iterations in (**e**).

surrogate models to select state points that are expected to be effective based on prior data; in essence, we expect the LLM to do the same using its internal representation of text data as the results are streamed to it. We provide a lower bound on performance using random sequence selection and evaluate the effect of a solution "seed" (i.e., a single known solution provided at the outset). We also performed repeated replicas for each algorithm and each target to quantify variance.

## Results

A representative rollout from the LLM optimizer is shown in Fig. 1e–g. In Fig. 1e, candidates proposed by the LLM in each iteration are shown in terms of distance *d* from the target, while in Fig. 1f, g, the same candidates are shown in the *Z*-space. This rollout shows how the LLM converges to the correct region within five iterations and then identifies many good solutions in the last five, balancing exploration and exploitation, sampling broadly first before successfully investigating minor variations.

Systematic benchmarking results are shown in Fig. 2. Representative rollouts for the membrane target are shown in Fig. 2a. The LLM is shown in two panels with different prompts: the "oracle" prompt, which provided no context about the process, and the "scientific" prompt, which provided details of the MD simulation being carried out (see Methods for full prompts). The LLM tends to converge more rapidly to a solution, while the other two progress more gradually, sampling a broader region. This mostly works in favor of the LLM, but can result in stagnating on bad solutions. In these rollouts, active learning failed to identify the correct region, while the evolutionary algorithm found several good solutions in the middle iterations but failed to exploit this before the end of the rollout. Note that each of the rollouts shown used the same initial random batch (darkest purple points in the plot).

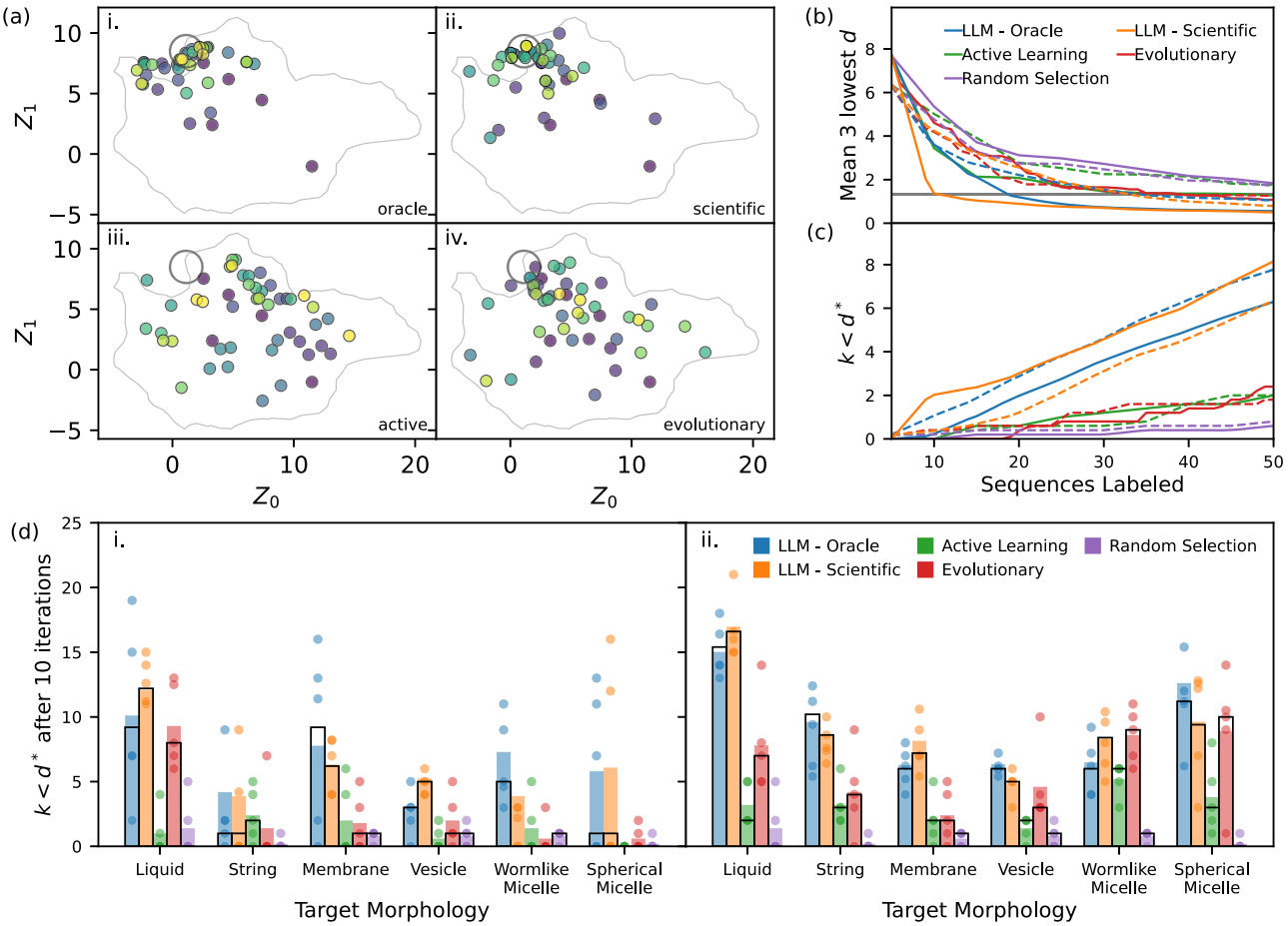


**Fig. 2 | Aggregated results of LLM-driven optimization experiments. a** Sample rollout for the "membrane" target for each of the search algorithms (i–iv). Points represent labeled samples, with lighter colors indicating candidates from later iterations (as with Fig. 1e–g). Circle indicates the 1.34 unit radius around the target, $Z_t$, representing accuracy due to stochastic self-assembly. **b** The mean distance from the target for the best three candidates during rollouts of the different optimization algorithms, lower values indicate better match. Solid lines are the mean of five replicas of the solution-seeded variant, while dashed lines are the mean of five replicas of the unseeded variant. Grey horizontal line indicates $d^* = 1.34$. **c** The number of sequences $k$ falling below the threshold score of $d^* = 1.34$ during rollouts of the different optimization algorithms. Larger values indicate more matches found. Same rollouts as panel e with alternate metric. **d** Aggregated performance of each algorithm on each target morphology (i) without and (ii) with solution seeds. Points are the results from each of the five replicas, empty bars (i.e., black outlines) are the median result, and filled bars are the mean result. Due to the stochastic nature of the LLM, replicas are repeated five times, and we report the median.

We show the best scores from each iteration in Fig. 2b. Since we are interested in discovering more than one candidate sequence, we consider the mean of the three lowest distances. While the quantitative results change with "top-$k$," trends are relatively insensitive to $k$. LLM-based schemes perform better than the others, achieving faster convergence and lower overall distance. Evolutionary algorithms and active learning are comparable in this case, and all algorithms perform better than random guessing.

Previously[5], we measured root mean squared deviation in $Z$ between identical replicas to be ~0.67 due to stochastic self-assembly. Therefore, we consider successful solutions to be about twice this distance from the target, $d^* = 1.34$. To quantify performance by the number of good solutions, we report $k < d^*$ as a function of labels in Fig. 2c, showing similar trends to Fig. 2b, with slight variations due to the discrete nature of the metric. In any case, LLM agents exhibit the best performance.

We repeated this for six target morphologies from ref. 4: liquid droplets, string-like aggregates, sheet-like membranes, hollow vesicles, wormlike micelles, and spherical micelles. The rollouts without solution seeding are shown in Fig. 2d–i. Substantial variation is observed between replicas and between targets, but some general trends emerge. This is a challenging problem, as each algorithm had a median $k = 0$ for at least one morphology. However, the LLM exhibits the highest mean performance in all six test cases. There was no clear winner between the scientific and oracle prompts for the LLM, with the oracle prompt giving a higher mean than the scientific prompt for three out of six morphologies.

For solution seeding, shown in Fig. 2d–ii, the performance typically increased compared to un-seeded rollouts, though not always (e.g., membranes). While mean $k$ increased only moderately in some cases, median $k$ increased substantially for the LLMs and the evolutionary algorithm; there were no median $k = 0$ cases when using seeding (note that we exclude the seed from being counted as a solution here). Both LLM prompts exhibited the highest mean performance for five of six morphologies, while they were competitive with the evolutionary algorithm in the last (wormlike micelles). The oracle and scientific prompts were evenly matched, with each prompt having the better mean performance for three of six morphologies (though not the same ones as in the un-seeded tests). Separately from LLM performance, we observed that the evolutionary algorithm benefited more from solution seeding than active learning, probably due to the continual re-use of relevant motifs as the solution seed remains in the mating pool throughout the rollout.

To evaluate the degree to which the Oracle LLM understood the task, it was given access to the message history from one representative rollout and asked to speculate on the nature of the function being optimized. The four tasks the LLM postulated were protein folding, polymer design, nanomaterial design, and drug design. When repeated for other rollouts, the first

guess was always protein folding (once called "Computational biology," but described as optimizing protein sequences). This raises the question of whether LLMs infer context even for "context-less" oracle-style prompts and subsequently adjust their behavior.

## Discussion

We demonstrated the effectiveness of LLMs in performing evolutionary optimization for the design of sequence-defined macromolecules. By comparing the performance of an LLM-based optimizer (Anthropic's Claude 3.5 Sonnet) with conventional active learning and evolutionary algorithms, we found that the LLM consistently outperformed the other methods in identifying monomer sequences that lead to the desired morphology.

Our results showed that the LLM-based optimizer often converged more rapidly to good solutions compared to the other approaches, seeming to strike a balance between exploration and exploitation. Both LLM strategies exhibited the best mean performance for 11 of the 12 test cases. We observed that the "oracle" prompt, which provided no context about the process, was equally as effective as the "scientific" prompt with or without solution seeding. We observed in some cases that the scientific prompt could guide the LLM to select good solutions sooner, but did not enhance its overall performance at the end of 10 iterations; this can be thought of as a bias towards certain solutions.

In all cases, LLM-driven optimization is roughly an order of magnitude better than random guessing, providing evidence for LLM reasoning. The LLM's performance in evolutionary optimization is notable, considering the concept was not explicitly trained. Matching the performance of the evolutionary algorithm indicates that the LLM follows a comparable process, without needing hyperparameters. While it is a stretch calling the LLM "parameter-free," it is controlled by natural language instructions, which may be more intuitive and nuanced than hyperparameters.

Despite our promising results, LLM-based optimization still faces some practical challenges[10]. For instance, the model does not always return candidates that fit the constraints (e.g., wrong composition, sequence duplication). There are also pathological cases where the model stops following instructions (e.g., claiming the problem is solved and no further candidates are required). While we have not performed systematic prompt engineering, it is clear that performance will be sensitive to prompt changes and even the specific language used.

Nevertheless, the prospect of LLM-driven evolutionary optimization is exciting and has already received significant attention.[7–10] The performance of these schemes should improve as the behaviors of LLMs are better understood, automated prompt engineering schemes are further refined, and reasoning capacity increases in each subsequent model generation. Notably, our scheme uses a widely available pre-trained LLM and requires no training data or examples. We expect LLM-driven optimization to play an expanding role in discovering and designing new materials, whether optimizing process conditions, chemistry, or any number of other design variables. To this end, future work should explore systematic prompt engineering, such as for enhanced consistency[15] and improved performance, reduction of bias in LLM responses as seen in our "scientific" prompt, as well as further investigation into how physical principles can be incorporated into—or removed from—LLM predictions.

## Methods

### Recurrent neural network

An RNN was used as the ground truth, or "oracle," as described in the LLM prompt. The pre-trained RNN was taken directly from our previous work[5] without modification. While not used directly in this work, we note that the training data for the RNN is publicly available[16]. Briefly, this is a bi-directional RNN with three GRU layers of hidden dimension seven, a fully connected linear layer, and ReLU activations throughout. The final predictions are the average from an ensemble of 10 individual models trained on one of 10 data folds.

Querying the RNN model instead of the MD simulation directly provides several advantages. First, the RNN is deterministic, while the MD simulations have significant stochasticity due to random initialization and thermal fluctuations, such that the same sequence under all the same conditions appears at an average distance of 1.6 units away. In the context of evolutionary optimization, this means the same sequences would need to be revisited more than once to be sure of the result. Furthermore, taking this RNN model as the ground truth permits orders of magnitude more tests of the evolutionary optimization performance since each MD simulation requires ~10 mins on an NVIDIA A100 GPU. Out of the $6 \times 10^4$ possible sequences with fixed composition or $5 \times 10^5$ with unconstrained composition, we previously evaluated $\sim 2 \times 10^3$ sequences[4]. In this work, we made $\sim 4 \times 10^4$ calls to the RNN model to produce the final results, which would require over 6000 GPU hours if MD simulations were performed instead.

While full verification of the method by MD is the subject of future work, we have performed a limited validation study and include the results in this article's Supplementary Information. However, we believe successful optimization of the RNN input by the LLM in this work is still interesting on its own, even without MD validation, as the RNN itself is a complex, highly nonlinear function that captures many of the trends in the simulation system[5]. It is this ability to optimize against a function (any function) not known to the LLM that we believe is the key result presented here.

### Performance metrics

For each candidate sequence $x_i$, the Euclidean distance $d_i$ between the $Z_i$ predicted by the pre-trained RNN model and the target $Z_t$ was computed. This $d_i$ is either reported directly, such as in Fig. 1e, or averaged, such as Fig. 2d. Elsewhere, the distance was converted to a binary outcome using a threshold distance $d^*$, and the number of sequences $k$ below this threshold was reported. This $k < d^*$ thus indicates the number of good sequences discovered during the rollout.

With each algorithm, we sample 10 batches of five sequences for 50 labeled sequences in total. Since this is highly variable, we repeat each test five times, which we refer to as replicas. To initialize rollouts, an initial batch of five sequences is randomly selected for each replica, with the same five sequences being used across all algorithms. These are counted as the first batch of five, so only nine batches are actually suggested by each algorithm.

### Active learning

We use an ensemble of Random Forest regressors as implemented in `scikit-learn`[17]. The binary sequence vector $x$ is the input, and the order parameter $Z$ is the output. Manual hyperparameter tuning yielded 16 estimators and no maximum depth. Final predictions are the mean of three such regressors, trained using three folds such that each model sees 2/3 of the total data during training; this permits uncertainty estimation for use in computing acquisition function. The acquisition function was Lower Confidence Bound $\alpha(x) = \mu(x) - \xi\sigma(x)$ with $\xi = 10^{-2}$, so the top five unlabeled sequences with the lowest Euclidean distance to $Z_t$, adjusted by a small consideration for unexplored or uncertain regions, were selected in each subsequent batch. After each iteration, the latest batch of $(x, Z)$ pairs was added to the labeled set, and the model was retrained on the expanded dataset. For parity with the other methods, all possible sequences with 12 $A$ and 8 $B$ were included, even though the RNN predictions are symmetric around the sequence.

### Evolutionary algorithm

We use the `eaSimple` implemented in the Distributed Evolutionary Algorithms Package (`DEAP`)[18,19]. The hyperparameters are as follows: There is a 50% chance to mate individuals by two-point crossover, followed by a 20% chance to mutate individuals, with each character having a 10% chance of a bit flip. Due to the nature of this generation method, we also penalize the individuals according to the penalty term $p = |N_B - 8|$, so

the fitness for an individual $i$ is,

$$F_i = -d_i - p_i = -||Z_i - Z_t|| - |N_{i,B} - 8|, \quad (1)$$

where $d_i$ is the Euclidean distance between the individual's $Z_i$ and the target $Z_t$, $N_{i,B}$ is the number of $B$-type monomers in the individual, and the negative sign leads to a minimization of the distance when the fitness is maximized. This soft penalty more closely matches the behavior of the LLM optimizer, which does not always produce sequences with exactly 12 $A$ and 8 $B$ monomers, although it does most of the time. Because some duplicate sequences are produced in each generation, we collect the first 50 sequences evaluated, rather than relying on the notion of batches as in the other methods.

### Random selection

We consider the random selection of sequences to be a lower performance bound for sequential learning. In each batch, we simply select five random sequences that have not yet been labeled. Note that the first batch is randomly selected but is the same for all the algorithms. When including solution seeding, we do not count the seed as a successful solution (as in all the algorithms).

### LLM interactions

The LLM performance tests were conducted using the Anthropic Claude 3.5 Sonnet model[11] using the Anthropic Python SDK[20]. The model is the base, commercially available model without fine-tuning, retrieval-augmented generation, or any other tools or examples. To minimize variance in responses, we set the `temperature` parameter to 0; note that this does not eliminate variance entirely. No system prompt was provided when making queries. Even beyond the stochasticity induced by finite temperature, LLMs are known to exhibit strong sensitivity to prompt detail[15,21,22], which means that the behavior is expected to change depending on the initial randomly selected batch. To investigate the variability, we conducted five replicas of each of the five different initial batches described elsewhere. Thus, the LLMs are called for a total of 25 times ($5 \times 5$), whereas the other methods, being deterministic, are only called five times. We report the median of each of these five replicas for comparison against the other algorithms. We note that the stochastic behavior of LLMs, especially proprietary ones such as Anthropic Claude, makes obtaining reproducible results more challenging. However, we believe that the observation of this phenomenon in refs. 7–10 lends credence to the results obtained here.

**Scientific prompt**. The following prompt was used to initialize the system (i.e., as the first user message):

> You are an evolutionary optimizer tasked with finding optimal sequences of 20 monomer beads, consisting of 12 attractive 'A' beads and 8 repulsive 'B' beads, that minimize the distance in order parameter space from the desired aggregate morphology in coarse-grained molecular dynamics (MD) simulations using the Kremer-Grest polymer model. The target morphology is `<description>`. The MD simulation takes a sequence of 20 monomers as input and calculates a numerical distance from the desired result using an order parameter. A smaller distance indicates that the self-assembled morphology is closer to the target. Please propose five sequences of 20 monomers in the format 'ABBA…', following these criteria: 1. Each sequence must be exactly 20 monomers long. 2. Each sequence must contain exactly 12 'A' monomers and 8 'B' monomers. When proposing new batches, consider the following guidelines: 1. Sequences with distances less than 1.34 are considered good solutions. Propose sequences similar to these, with variations to explore the nearby solution space. 2. Balance exploitation of promising solutions with exploration of new regions to avoid getting stuck in local minima. 3. Avoid proposing duplicate or previously evaluated sequences. 4. The optimization process will continue for 10 iterations. Plan your strategy accordingly to identify as many good solutions as possible. 5. For each batch, provide chain-of-thought reasoning to justify your proposed sequences, considering the following factors: a. The influence of monomer sequence on the resulting morphology is based on the Kremer-Grest polymer model and the principles of self-assembly. b. The role of attractive ('A') and repulsive ('B') interactions in determining the aggregate structure. c. Insights gained from previous iterations, including patterns or motifs that lead to lower distances. d. The need to explore diverse sequences while also refining promising solutions. Remember to maintain a clear and concise format for your proposed sequences and explanations.

After each iteration, the following user message was returned to the model:

> Here are the sequences from the previous batch and their corresponding distances to the desired morphology:
>
> `<result>`
>
> Note that only distances of <1.34 are considered good solutions. Please propose five new 20-monomer sequences (12 'A', 8 'B') that build upon the insights from these results to yield structures closer to the target: `<description>`. Remember to: 1. Balance exploitation and exploration. 2. Avoid proposing duplicate or previously evaluated sequences. 3. Provide a brief rationale for your proposals. 4. Apply principles from the polymer self-assembly context.

The `<result>` token in the above was replaced with a series of "`<sequence>: <distance>`" entries:

```
AAAAABAABAAABAAAABAB: 9.076

AAABAAABAAAABAAABBAB: 8.565

AAABAABAABAAABAAABAB: 10.828

AAABAABAABAAABAABABA: 12.924

AAABAABAABAAABAABAAB: 12.061
```

The ordering of these was the same as the order proposed by the LLM assistant, and the distances were obtained from the RNN surrogate model as described above.

**Target morphologies**. For the prompts with scientific context, a description of the target morphology was provided in addition to the task description. These natural language descriptions are shown in Table 1. The sequences were chosen according to their relevance in our prior work[4,5]. The structures are rendered in Fig. 1b and can be studied in greater detail via our data repository[16].

**Oracle prompt**. The following prompt was used to initialize the system (i.e., as the first user message):

> You are an evolutionary optimizer tasked with finding optimal sequences of 20 characters, consisting of 12 'A' characters and 'B' characters that minimizes a metric. I have an oracle that takes in the sequence of 20 characters and produces a numerical distance from the desired result. The smaller the distance, the closer the sequence is to the desired result. Please propose five sequences of 20 characters in the format 'ABBA…', following these criteria: 1. Each sequence must be exactly 20 characters long. 2. Each sequence must contain exactly 12 'A' characters and 8 'B' characters. When proposing new batches,

**Table 1 | Target morphologies with their sequences and natural language descriptions**

| Morphology | Sequence | Natural language description |
|---|---|---|
| Liquid | BAABAABAABAABABBAAAB | amorphous liquid droplets |
| String | ABBABABAAAAABABBABAA | string-like aggregates |
| Membrane | ABAAABBAAAABBABBBAAA | sheet-like membranes |
| Vesicle | BABBBAABAAAABAABAABA | hollow vesicles |
| Wormlike micelle | BBAAABBBAAAABBBAAAAA | heterogeneous, worm-like aggregates |
| Spherical micelle | BBBBBBBBAAAAAAAAAAAA | uniform spherical micelles |

consider the following guidelines: 1. Sequences with distances less than 1.34 are considered good solutions. Propose sequences similar to these, with variations to explore the nearby solution space. 2. Balance exploitation of promising solutions with exploration of new regions to avoid getting stuck in local minima. 3. Avoid proposing duplicate or previously evaluated sequences. 4. The optimization process will continue for 10 iterations. Plan your strategy accordingly to identify as many good solutions as possible. 5. For each batch, provide chain-of-thought reasoning to justify your proposed sequences, considering the following factors: a. The influence of character sequence on the resulting distance metric, based on what you have observed. b. The role of 'A' and 'B' type characters in determining the resulting distance. c. Insights gained from previous iterations, including patterns or motifs that lead to lower distances. d. The need to explore diverse sequences while also refining promising solutions. Remember to maintain a clear and concise format for your proposed sequences and explanations.

After each iteration, the following user message was returned to the model:

Here are the sequences from the previous batch and their corresponding distances to the desired result:

`<result>`

Note that only distances of less than 1.34 are considered good solutions. Please propose five new 20-character sequences (12 'A', 8 'B') that build upon the insights from these results to yield distances closer to the target. Remember to: 1. Balance exploitation and exploration. 2. Avoid proposing duplicate or previously evaluated sequences. 3. Provide a brief rationale for your proposals.

Note the only difference between this and the scientific prompt above was the use of the term "character" instead of "monomer" and the omission of the final parenthetical ("more like…").

**Seeded prompts**. In the case of the "seeded" experiments, the exact target solution was given after the base prompt, using the following language:

Hint: I already know that `<sequence>` gives a good result.

**Initial batch**. Both types of prompts included a random initial batch of sequences (regardless of solution seeding). These were the sequences provided to the other optimization algorithms (active learning, evolutionary, and random selection) to mitigate the effect of initialization in comparing the different algorithms. In the case of the two LLM strategies, this information was included after the main prompt like so:

Here is an initial batch selected completely at random to get you started: `<result>`

### Error correction

Not every sequence proposed by the LLM was suitable for labeling. While most proposed sequences conformed to the rules described in the prompt, some proposals were not exactly 20 characters long. Furthermore, the LLM did not always propose unseen sequences in later batches. Rather than permit repeated labeling of the same sequences or permitting more than 10 iterations, which would make the comparison to other algorithms much more complicated, we implemented a simple error correction scheme:

- Reject sequences with the wrong length
- Among accepted sequences, perform random bit flips until a new sequence is discovered
- Repeat Step 2 until five new sequences are proposedWhile the LLM does not perfectly follow the instructions (e.g., avoid proposing duplicate sequences), including randomly mutated sequences should provide a lower bound on performance, with better instruction-following should increase performance via more intentional candidate selection.

We considered an alternative scheme where the optimization rollout could continue until reaching a labeling budget. This resulted in unpredictable behaviors, including infinite loops where the candidate budget was never reached before time-out (while incurring substantial costs against our project budget) and early termination where the model would refuse to produce any more sequences, claiming that a sufficient number of suitable candidates had already been found. Both scenarios complicated the comparisons to the point of being inconclusive. We speculate that more sophisticated prompt engineering could be used to avoid these behaviors in the future.

### Oracle task speculation

The LLM was given access to the history of the first replica of the spherical micelle optimization rollout using the Oracle prompt without solution seeding. A new user prompt was appended to the bottom to inquire about the possible nature of the oracle function. The model usually returned something like this:

**User:** Based on the conversation, please speculate on the nature of the oracle function described above.

**Assistant:** Based on the conversation, it appears that the oracle function is evaluating the similarity or closeness of the proposed 20-character sequences to a target sequence. The function takes a sequence of 'A' and 'B' characters as input and returns a numerical distance value, where a smaller distance indicates a closer match to the desired result. Some key observations about the oracle function:

- The target sequence likely consists of a specific arrangement of 'A' and 'B' characters, and the goal is to find sequences that closely match this target.
- The function seems to be sensitive to the position and order of the characters in the sequence, as small changes in the arrangement can lead to significant differences in the distance value.
- The optimal solution appears to have approximately 12 'A' characters and 8 'B' characters, suggesting that the target sequence has a similar composition.
- The distance values returned by the function are non-negative, with a value less than 1.34 considered a good solution.
- The function is deterministic, meaning that it will always return the same distance value for a given input sequence.

Based on these observations, …

After the ellipsis, the message would continue with some guesses about the function itself. For instance, the following is a continuation of the exact message above:

**Table 2 | Summary of responses when the LLM is asked to speculate about the nature of the computer simulation it is being asked to optimize for**

| Morphology | First task listed in output |
|---|---|
| Liquid | Protein folding: the 'A' and 'B' characters might represent different amino acids, and the optimal sequence could be the one that results in a stable, functional protein structure. |
| String | Computational biology: optimizing DNA or protein sequences for specific properties. |
| Membrane | Protein folding: finding the optimal 3D structure of a protein based on its amino-acid sequence. |
| Vesicle | Protein folding: the sequence could represent a simplified model of a protein, and the simulation is trying to find the most stable configuration. |
| Wormlike micelle | Protein folding: the 'A' and 'B' characters might represent different amino acids, and the simulation is trying to find the optimal sequence that minimizes the energy of the folded protein structure. |
| Spherical micelle | Protein folding: optimizing the sequence of amino acids (represented by 'A' and 'B') to minimize the energy state of a protein. |

**Assistant:** … the oracle function is likely a string similarity metric, such as Levenshtein distance or Hamming distance, which measures the difference between two strings. The function compares the input sequence to a predetermined target sequence and calculates the distance based on the number and position of character differences.

In the remainder of this section, only the User prompt and the latter portion of the Assistant response will be shown for brevity, as the intermediate portion is nearly identical in each case. As the task description becomes more detailed, the LLM guesses more accurately. Below we highlight the additional detail provided in bold and show how the response changes:

**User: The oracle function referred to above is a computer simulation**. Based on the conversation, please speculate on the nature of this **computer simulation**.

**Assistant:** …the simulation could be related to various fields, such as:

- Protein folding: The sequence of 'A' and 'B' characters might represent different amino acids, and the simulation could be trying to find a specific protein structure that minimizes the free energy.
- Combinatorial optimization: The problem could be an abstracted version of a real-world optimization task, such as finding the shortest path in a network or the optimal arrangement of components in a system.
- Cryptography: The simulation might be related to breaking a simple substitution cipher, where 'A' and 'B' represent encrypted characters, and the goal is to find the correct decryption key.

Without more context about the specific problem domain, it is difficult to determine the exact nature of the simulation.

In the above, the response gets much closer to the true nature of the MD simulation, selecting protein folding as the top choice. Finally, we narrow down the possibility space to computer simulations in the physical sciences:

**User**: The oracle function referred to above is a computer simulation **used in the physical sciences**. Based on the conversation, please speculate on the nature of this computer simulation.

**Assistant:** … the simulation seems to be related to a physical science problem, possibly in the fields of molecular biology, materials science, or chemical engineering. Some potential applications could be:

1. Protein folding: The 'A' and 'B' characters might represent different amino acids, and the optimal sequence could be the one that results in a stable, functional protein structure.

2. Polymer design: The characters could represent different monomers, and the simulation might be trying to find the optimal sequence that gives the polymer specific desired properties.

3. Nanomaterial design: The characters could represent different atomic or molecular components, and the simulation could be searching for the optimal arrangement that yields specific electronic, optical, or mechanical properties.

4. Drug design: The characters might represent different chemical groups, and the simulation could be looking for the optimal drug molecule that binds effectively to a specific target.

With this change, the exact problem of monomer sequence selection for polymer design is proposed by the LLM. We repeat this last prompt for the first replica of each morphology target and report only the first result in Table 2.

## Data availability

Our data are publicly available on Zenodo[23].

## Code availability

Our code is publicly available on Zenodo[23].

Received: 17 May 2024; Accepted: 21 October 2024;
Published online: 18 November 2024

## References

1. Cummins, C. et al. Enabling future nanomanufacturing through block copolymer self-assembly: a review. *Nano Today* **35**, 100936 (2020).
2. Dill, K. A. & MacCallum, J. L. The protein-folding problem, 50 years on. *Science* **338**, 1042–1046 (2012).
3. Hakobyan, K., Noble, B. B. & Xu, J. The current science of sequence-defined macromolecules. *Prog. Polym. Sci.* **147**, 101754 (2023).
4. Statt, A., Kleeblatt, D. C. & Reinhart, W. F. Unsupervised learning of sequence-specific aggregation behavior for a model copolymer. *Soft matter* **17**, 7697–7707 (2021).
5. Bhattacharya, D., Kleeblatt, D. C., Statt, A. & Reinhart, W. F. Predicting aggregate morphology of sequence-defined macromolecules with recurrent neural networks. *Soft Matter* **18**, 5037–5051 (2022).
6. Statt, A., Casademunt, H., Brangwynne, C. & Panagiotopoulos, A. Model for intrinsically disordered proteins with a strong dependence of liquid-liquid phase separation on sequence. *In*: Bulletin of the American Physical Society 65 (2020).
7. Wu, X., Wu, S.-h., Wu, J., Feng, L. & Tan, K. C. Evolutionary computation in the era of large language model: Survey and roadmap. *arXiv* https://arxiv.org/abs/2401.10034 (2024).
8. Yang, C. et al. Large language models as optimizers. *arXiv* https://arxiv.org/abs/2309.03409 (2023).

9. Liu, F. et al. Large language model for multi-objective evolutionary optimization. *arXiv* https://arxiv.org/abs/2310.12541 (2023).
10. Brahmachary, S. et al. Large language model-based evolutionary optimizer: reasoning with elitism. *arXiv* https://arxiv.org/abs/2403.02054 (2024).
11. Anthropic. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet (2024).
12. Kusne, A. G. et al. On-the-fly closed-loop materials discovery via bayesian active learning. *Nat. Commun.* **11**, 5966 (2020).
13. Kim, C., Batra, R., Chen, L., Tran, H. & Ramprasad, R. Polymer design using genetic algorithm and machine learning. *Computational Mater. Sci.* **186**, 110067 (2021).
14. Abolhasani, M. & Kumacheva, E. The rise of self-driving labs in chemical and materials sciences. *Nat. Synth.* **2**, 483–492 (2023).
15. Wang, L. et al. Prompt engineering in consistency and reliability with the evidence-based guideline for llms. *npj Digital Med.* **7**, 41 (2024).
16. Bhattacharya, D., Kleeblatt, D., Statt, A. & Reinhart, W. Data for "Predicting aggregate morphology of sequence-defined macromolecules with Recurrent Neural Networks" (2022).
17. Pedregosa, F. et al. Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011).
18. Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M. & Gagné, C. DEAP: Evolutionary algorithms made easy. *J. Mach. Learn. Res.* **13**, 2171–2175 (2012).
19. Bäck, T. Evolutionary computation 1: Basic algorithms and operators (CRC press, 2018).
20. Anthropic. Anthropic python API library. https://github.com/anthropics/anthropic-sdk-python (2024).
21. Sclar, M., Choi, Y., Tsvetkov, Y. & Suhr, A. Quantifying language models' sensitivity to spurious features in prompt design or: how i learned to start worrying about prompt formatting. *arXiv* https://arxiv.org/abs/2310.11324 (2023).
22. Errica, F., Siracusano, G., Sanvito, D. & Bifulco, R. What did i do wrong? quantifying llms' sensitivity and consistency to prompt engineering. *arXiv* https://arxiv.org/abs/2406.12334 (2024).
23. Reinhart, W. & Statt, A. Data and code for, "Large language models design sequence-defined macromolecules via evolutionary optimization" https://doi.org/10.5281/zenodo.13489838 (2024).


## Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. DMR-2401663 and DMR-2401664.We acknowledge the Pennsylvania State University Institute for Computational and Data Sciences (ICDS) for computational resources and financial support and the Department of Materials Science and Engineering for financial support.

## Author contributions

W.F.R. designed the study, collected the data, performed analysis, and provided resources for the work. W.F.R. and A.S. acquired funding, interpreted the results, and wrote the article together.

## Competing interests

The authors declare no competing interests.

## Additional information

**Supplementary information** The online version contains supplementary material available at https://doi.org/10.1038/s41524-024-01449-6.

**Correspondence** and requests for materials should be addressed to Wesley F. Reinhart or Antonia Statt.