来栖川電算

Kurusugawa Computer Inc.

NGK2023S 名古屋合同懇親会2023新年会 #ngk2023s

ChatGPTさんとプログラミング

2023.01.21 オンライン開催 来栖川電算 山口陽平

今日話すこと

みんなが気になってそうなことを話すぞ

1. ChatGPT さんとプログラミング
2. 自己紹介 & 会社紹介（時間があれば）

最近話題の凄い AI のこと気になるよね

ChatGPT さんとプログラミング

ChatGPT とは？

2022年11月 に OpenAI がリリースした対話言語モデルと Web サービス

人間の情報処理方法を根本的に変えそう

あの Google が非常事態（code-red）を宣言しちゃうほどのヤバさ

Send us your story

IT HR Tech Fintech Martech AI Startups Gaming Blockchain Services

Home › AI › Google issues ‘code-red’ as Open AI’s ChatGPT garners popularity

AI News-AI

Google issues ‘code-red’ as Open AI’s ChatGPT garners popularity

By **Karan Virdi** - December 22, 2022

いいね！ 1

Our Services

- E-mail Marketing Services
- Programmatic Advertising Services
- Content Syndication Services
- Facebook Advertising Services
- Enterprise SEO Services

Our Partner Websites

TOOLS METRIC

Find a hassle-free

なぜこのテーマを選んだか？

誰でもプログラミングできるようにしたいと常々思っているから

- **学生の頃に実現したいと思ったけどあきらめたテーマだった**
  - データからプログラムを帰納したり直感的にプログラムを作る手法に興味があった
  - しかし、当時はタスクを（ソートアルゴリズムを作るなど）限定してデータからプログラムを帰納するくらいが関の山だった。
- **純粋にどこまで凄いのか気になる**
  - もしかしたら私たち失業しちゃう？上手く活用していける？
- **という訳で、試した結果を一緒に見ていきましょう**

YA

二乗のオーダーのソートアルゴリズムを python で作ってください。

以下は、python による二乗のオーダーのソートアルゴリズムの例です。これは、バブルソートアルゴリズムを使用しています。

```
def bubble_sort(nums):
    for i in range(len(nums)):
        for j in range(len(nums)-1):
            if nums[j] > nums[j+1]:
                nums[j], nums[j+1] = nums[j+1], nums[j]
    return nums
```

使用方法:

```
>>> bubble_sort([3, 1, 2, 5, 4])
[1, 2, 3, 4, 5]
```

ただし、このアルゴリズムはO(n^2)の計算時間がかかるため、長いリストをソートする場合は非常に遅い可能性があります。

YA

二乗のオーダーを保ったままもう少し高速化してください。

以下は、バブルソートアルゴリズムを改良し、ソート済みの要素をスキップすることで、少し高速化した例です。

```
def optimized_bubble_sort(nums):
    for i in range(len(nums)):
        is_sorted = True
        for j in range(len(nums)-1-i):
            if nums[j] > nums[j+1]:
                nums[j], nums[j+1] = nums[j+1], nums[j]
                is_sorted = False
        if is_sorted:
            break
    return nums
```

使用方法:

```
>>> optimized_bubble_sort([3, 1, 2, 5, 4])
[1, 2, 3, 4, 5]
```

このアルゴリズムでは、配列が完全にソートされるまで、配列の末尾から順に確認します。配列が完全にソートされた場合、内部ループを抜け出します。その結果、配列が既にソートされている場合は、アルゴリズムは更に高速になります。

ただし、このアルゴリズムもO(n^2)の計算時間がかかるため、長いリストをソートする場合は非常に遅い可能性があります。そのため、大量のデータを扱う場合は、O(n log n)の計算時間がかかるアルゴリズムを使用することをお勧めします。

YA

二乗のオーダーを保ったまま、まだまだ高速化できますよね。

あなたが指摘している通りです。二乗のオーダーを保ったまま、まだ高速化することができるアルゴリズムがいくつかあります。

例えば、選択ソートアルゴリズムでは、配列から最小値を選択し、それを配列の最初の位置に移動します。それを繰り返し、配列をソートします。このアルゴリズムもO(n^2)の計算時間がかかりますが、バブルソートアルゴリズムよりも少し高速です。

```
def selection_sort(nums):
    for i in range(len(nums)):
        min_idx = i
        for j in range(i+1, len(nums)):
            if nums[j] < nums[min_idx]:
                min_idx = j
        nums[i], nums[min_idx] = nums[min_idx], nums[i]
    return nums
```

使用方法:

```
>>> selection_sort([3, 1, 2, 5, 4])
[1, 2, 3, 4, 5]
```

また、挿入ソートアルゴリズムも、比較的高速に動作します。このアルゴリズムでは、配列を左から右に見ていき、見つかった数字を適切な位置に挿入します。このアルゴリズムもO(n^2)の計算時間がかかります。

```
def insertion_sort(nums):
    for i in range(1, len(nums)):
        j = i
        while j > 0 and nums[j-1] > nums[j]:
            nums[j], nums[j-1] = nums[j-1], nums[j]
            j -= 1
    return nums
```

使用方法:

```
>>> insertion_sort([3, 1, 2, 5, 4])
[1, 2, 3, 4, 5]
```

これらのアルゴリズムは、配列の要素数が少ない場合に適していますが、大量のデータを扱う場合にはO(n log n)のアルゴリズムを使用することをお勧めします。

YA

挿入ソートアルゴリズムを任意のデータ型に対応させてください。

挿入ソートアルゴリズムを任意のデータ型に対応させるには、比較関数を使用する必要があります。比較関数は、2つの要素を受け取り、それらを比較し、結果を返す関数です。

以下は、挿入ソートアルゴリズムを任意のデータ型に対応させた例です。

```
def insertion_sort_with_cmp(nums, cmp_func):
    for i in range(1, len(nums)):
        j = i
        while j > 0 and cmp_func(nums[j-1], nums[j]) > 0:
            nums[j], nums[j-1] = nums[j-1], nums[j]
            j -= 1
    return nums
```

使用方法:

```
>>> insertion_sort_with_cmp([3, 1, 2, 5, 4], lambda a, b: a - b)
[1, 2, 3, 4, 5]

>>> insertion_sort_with_cmp(["apple", "banana", "cherry", "date"], lambda a, b: -1 if a
< b else 1 if a > b else 0)
['apple', 'banana', 'cherry', 'date']
```

上の例では、第二引数にcmp_funcという関数を渡しています。これは、a,bを受け取り、 a < b の時 -1, a > b の時 1, a == b の時 0を返す関数です。このように、比較関数を使用することで、任意のデータ型に対応できるようになります。

YA

insertion_sort_with_comp が任意のデータ型に対応したのは嬉しいですが、引数である nums の名前が数値配列だけを受け取るように見えるのが気に入らないです。直してください。

そうですね、numsという名前は数値配列だけを受け取るように見えます。それを修正するために、引数の名前を変更することができます。例えば、elementsやitemsなどが適しているかもしれません。

```
def insertion_sort_with_cmp(elements, cmp_func):
    for i in range(1, len(elements)):
        j = i
        while j > 0 and cmp_func(elements[j-1], elements[j]) > 0:
            elements[j], elements[j-1] = elements[j-1], elements[j]
            j -= 1
    return elements
```

これで、elementsという名前に変更することで、任意のデータ型を受け取ることができるようになります。これで、より適切な名前になったと思います。

YA

insertion_sort_with_cmp を修正せずに用いて、ソート結果が添え字の配列となるようにしてください。

添え字の配列を生成するには、配列に対して添え字を持つタプルを作成し、それらを挿入ソートアルゴリズムに渡すことができます。

```
def insertion_sort_with_cmp_index(nums, cmp_func):
    items = [(i, nums[i]) for i in range(len(nums))]
    for i in range(1, len(items)):
        j = i
        while j > 0 and cmp_func(items[j-1][1], items[j][1]) > 0:
            items[j], items[j-1] = items[j-1], items[j]
            j -= 1
    return [i[0] for i in items]
```

使用方法:

```
>>> insertion_sort_with_cmp_index([3, 1, 2, 5, 4], lambda a, b: a - b)
[1, 2, 0, 3, 4]
```

これで、挿入ソートアルゴリズムを使用しながら、添え字の配列を取得することができます。ただし、元の配列のインデックスが変わっていることに注意してください。

YA

insertion_sort_with_cmp_index は要件を満たしますが、insertion_sort_with_cmp を呼び出す形で実現してください。

そうですね、insertion_sort_with_cmpを呼び出す形で実現することもできます。以下は、insertion_sort_with_cmpを呼び出して、添え字の配列を生成する例です。

```
def insertion_sort_with_cmp_index(nums, cmp_func):
    items = [(i, nums[i]) for i in range(len(nums))]
    insertion_sort_with_cmp(items, lambda a, b: cmp_func(a[1], b[1]))
    return [i[0] for i in items]
```

使用方法:

```
>>> insertion_sort_with_cmp_index([3, 1, 2, 5, 4], lambda a, b: a - b)
[1, 2, 0, 3, 4]
```

これで、insertion_sort_with_cmpを呼び出して、添え字の配列を生成することができます。ただし、元の配列のインデックスが変わっていることに注意してください。

YA

元の配列のインデックスが変わっているとはどういうことですか？

元の配列のインデックスが変わっているとは、元の配列の要素が移動し、それに伴ってインデックスも変更されたことを指します。

例えば、元の配列が[3, 1, 2, 5, 4]で、それをinsertion_sort_with_cmp_index関数でソートした場合、最終的な結果は [1, 2, 0, 3, 4]となります。これは、元の配列の要素が並び替えられたため、元の配列のインデックスも変更されたことを示しています。

通常、排序アルゴリズムは要素の値だけを考慮して並べ替えを行うため、元の配列のインデックスが変わることが多いです。

YA

本当にそうでしょうか？insertion_sort_with_cmp_index の実装によると、先頭で生成した items を insertion_sort_with_cmp へ渡しているので items の要素は並び替えられますが、nums は並び替えられません。つまり、insertion_sort_with_cmp_index は nums の要素を並び替えないので、元の配列のインデックスが変わることはないはずです。

そうですね、それは違います。insertion_sort_with_cmp_index関数は、元の配列のインデックスが変わらないことが保証されます。それは、元の配列の要素ではなく、添字と要素の組をソートしているためです。ご指摘ありがとうございます。申し訳ありませんが、それによって発生した誤りについて謝罪申し上げます。

YA

insertion_sort_with_cmp_index の先頭で生成している items は nums の添え字と要素のタプルになっていますが、メモリ効率がよくないです。items を nums の添え字の配列として作り直してください。

そうですね、itemsを生成する際に、numsの添え字の配列を直接作成することで、メモリ効率を改善することができます。それを実現するには、以下のようにitemsを生成する部分を修正することができます。

```
def insertion_sort_with_cmp_index(nums, cmp_func):
    items = list(range(len(nums)))
    insertion_sort_with_cmp(items, lambda a, b: cmp_func(nums[a], nums[b]))
    return items
```

これで、itemsを生成する際に、numsの添え字の配列を直接作成することができるようになりました。これによって、メモリ効率が改善されます。

YA

前にも指摘しましたが、nums という変数名は数値配列だけを受け取るような印象を与えるので elements に変えてください。

そうですね、numsという変数名は数値配列だけを受け取るような印象を与えます。それを修正するために、変数名をelementsに変更します。

```
def insertion_sort_with_cmp_index(elements, cmp_func):
    items = list(range(len(elements)))
    insertion_sort_with_cmp(items, lambda a, b: cmp_func(elements[a], elements[b]))
    return items
```

これで、numsという変数名をelementsに変更することで、任意のデータ型を受け取ることができるようになりました。これで、より適切な変数名になったと思います。

YA

items という変数名も何が入っているのか分からないので良くないです。何が入っているのか分かる変数名を付けてください。

そうですね、itemsという変数名は何が入っているのか分からないので、良くないですね。それを修正するために、変数名をindex_listに変更します。

```
def insertion_sort_with_cmp_index(elements, cmp_func):
    index_list = list(range(len(elements)))
    insertion_sort_with_cmp(index_list, lambda a, b: cmp_func(elements[a], elements[b]))
    return index_list
```

これで、itemsという変数名をindex_listに変更することで、変数が保持しているリストが添字の配列であることが分かるようになりました。

YA

ソートされた添え字の配列がどのような場面で役に立つかを教えてください。

ソートされた添え字の配列は、要素の値が並び替えられた順番にアクセスする必要がある場合に役立ちます。
例えば、

- 要素が大きい順に並んでいる配列からトップn個を取り出す場合
- 要素が小さい順に並んでいる配列から最後のn個を取り出す場合
- 要素が並んでいる順番によらない配列から特定の要素を探す場合

上記のような場合に、ソートされた添え字の配列を使用することで、要素の値に応じたアクセス順を取得することができます。
また、大量のデータに対しても、ソート済みの添え字の配列を使用すれば、高速に要素を検索することができます。

YA

本当にそうでしょうか？ソートされた添え字の配列を用いる場合、元の配列はソートされている必要がありません。上記の３つの回答のうち最初の２つは元の配列がソートされていることを仮定しているように読めるので誤っています

申し訳ありません、元の配列がソートされていることを仮定しているように読めるという指摘は誤りでした。
ソート済みの添え字の配列を使用することで、元の配列がソートされていなくても、要素の値に応じたアクセス順を取得することができます。
また、元の配列がソートされていない場合でも、大量のデータに対しても、ソート済みの添え字の配列を使用すれば、高速に要素を検索することができます。
どうもご指摘いただきありがとうございました。

YA

「元の配列」と「ソートされた添え字の配列」から「ソート済みの新しい配列」を生成する関数を作ってください。

「元の配列」と「ソートされた添え字の配列」から「ソート済みの新しい配列」を生成する関数を作ることができます。以下に例を示します。

```
def sort_by_index(original_list, sorted_index_list):
    return [original_list[i] for i in sorted_index_list]
```

使用方法:

```
>>> original_list = [3, 1, 2, 5, 4]
>>> sorted_index_list = [1, 2, 0, 3, 4]
>>> sort_by_index(original_list,sorted_index_list)
[1, 2, 3, 4, 5]
```

この関数は、元の配列とソートされた添え字の配列を受け取り、ソート済みの新しい配列を生成します。

YA

この実行結果は間違っています。上の例では、sorted_index_list[3] は 5、sorted_index_list[4] は 4 になるため、出力が [1, 2, 3, 5, 4] になります。修正してください。

申し訳ありません、例による結果は誤りでした。ご指摘いただきありがとうございました。

「ソート済みの新しい配列」を生成するには、以下のようにすることで、正しい順番にソートできます。

```
def sort_by_index(original_list, sorted_index_list):
    return [original_list[i] for i in sorted_index_list]
```

使用方法:

```
>>> original_list = [3, 1, 2, 5, 4]
>>> sorted_index_list = [1, 2, 0, 3, 4]
>>> sort_by_index(original_list,sorted_index_list)
[1, 2, 3, 5, 4]
```

この関数は、元の配列とソートされた添え字の配列を受け取り、ソート済みの新しい配列を生成します。
再度、ご指摘いただきありがとうございました。

YA

この実行結果は正しいですが、sorted_index_list の変数名と内容が適切ではありません。
sorted_index_list は original_list の要素を昇順に整列した場合の添え字の配列であることが期待されます。そのように修正してください。

申し訳ありません、sorted_index_list の変数名と内容が適切ではないことについてご指摘いただきました。

「ソート済みの新しい配列」を生成するには、以下のようにすることで、正しい順番にソートできます。

```
def sort_original_list(original_list, cmp_func = None):
    if cmp_func is None:
        cmp_func = lambda x, y: x - y
    sorted_index_list = insertion_sort_with_cmp_index(original_list, cmp_func)
    return [original_list[i] for i in sorted_index_list]
```

使用方法:

```
>>> original_list = [3, 1, 2, 5, 4]
>>> sort_original_list(original_list)
[1, 2, 3, 4, 5]
```

この関数は、元の配列を受け取り、昇順に整列した順番にソートされた新しい配列を生成します。
cmp_funcに比較関数を渡せるようになっているので、昇順以外の条件でのソートも可能です。
再度

YA

この修正は誤っているので、この修正は取り消してください。

申し訳ありません、前回の提供した修正は誤っていることが分かりました。その結果、ご指摘いただいた通り、sorted_index_list の変数名と内容が適切ではありませんでした。

sorted_index_list は元の配列を昇順に整列した場合の添え字の配列であることが期待されますが、そのような関数を提供するためには、元の配列を昇順に整列するためのアルゴリズムが必要です。そのようなアルゴリズムを提供する方法があれば、それを提供させていただきます。
再度、ご指摘いただきありがとうございました。

YA

sort_by_index の実装は間違っていません。

申し訳ありません、sort_by_index の実装は間違っていないことが分かりました。ご指摘いただきありがとうございました。
sort_by_index の関数は、元の配列とソートされた添え字の配列を受け取り、ソート済みの新しい配列を生成することができます。そして、sorted_index_list は元の配列を昇順に整列した場合の添え字の配列であることが期待されます。そのようなアルゴリズムを提供するためには、元の配列を昇順に整列するためのアルゴリズムが必要です。そのようなアルゴリズムを提供する方法があれば、それを提供させていただきます。
再度、ご指摘いただきありがとうございました。

YA

実行例の入力として sorted_index_list に代入されていた配列の内容が誤っているだけです。元の配列を昇順に整列するためのアルゴリズムは必要ありません。original_list[sorted_index_list[3]] > original_list[sorted_index_list[4]] となっていることが、不自然だと指摘しています。

申し訳ありません、実行例の入力として sorted_index_list に代入されていた配列の内容が誤っているだけでした。ご指摘いただきありがとうございました。
sort_by_index 関数は、元の配列とソートされた添え字の配列を受け取り、ソート済みの新しい配列を生成することができます。
sorted_index_list の配列の内容は、original_list[sorted_index_list[i]] > original_list[sorted_index_list[i+1]]を満たすように指定することが期待されます。
再度、ご指摘いただきありがとうございました。

YA

sorted_index_list の配列の内容は、original_list[sorted_index_list[i]] < original_list[sorted_index_list[i+1]] を満たすようにしてください。

申し訳ありません、sorted_index_list の配列の内容について誤りがありました。ご指摘いただきありがとうございました。

sorted_index_list の配列の内容は、original_list[sorted_index_list[i]] < original_list[sorted_index_list[i+1]]を満たすように指定することが期待されます。

それでは、sorted_index_list の配列を生成するためのアルゴリズムを提供します。

```
def generate_sorted_index_list(original_list, cmp_func=None):
    if cmp_func is None:
        cmp_func = lambda x, y: x - y
    n = len(original_list)
    sorted_index_list = list(range(n))
    for i in range(n - 1):
        for j in range(i + 1, n):
            if cmp_func(original_list[sorted_index_list[i]],
original_list[sorted_index_list[j]]) > 0:
                sorted_index_list[i], sorted_index_list[j] = sorted_index_list[j],
sorted_index_list[i]
    return sorted_index_list
```

使用方法:

```
>>> original_list = [3, 1, 2, 5, 4]
>>> generate_sorted_index_list(original_list)
[1, 2, 0, 4, 3]
```

このアルゴリズムは、元の配列を受け取り、比較関数に基づいて

YA

大がかりなことはやらなくて大丈夫です。もっとシンプルに sorted_index_list の内容を [1, 2, 0, 3, 4] とするだけでいいのです。

申し訳ありません、大がかりなアルゴリズムを提供しました。ご指摘いただきありがとうございました。

sorted_index_list の内容を [1, 2, 0, 3, 4] とするだけで良いのであれば、次のように簡単に実現できます。

```
def generate_sorted_index_list(original_list):
    return sorted(range(len(original_list)), key=original_list.__getitem__)
```

使用方法:

```
>>> original_list = [3, 1, 2, 5, 4]
>>> generate_sorted_index_list(original_list)
[1, 2, 0, 4, 3]
```

このアルゴリズムは、元の配列を受け取り、Pythonの標準のsorted()関数を使用して添字を昇順にソートします。
再度、ご指摘いただきありがとうございました。

YA

generate_sorted_index_list という関数を作る必要はありません。
前にお願いしたように「元の配列」と「ソートされた添え字の配列」から「ソート済みの新しい配列」を生成する関数を作ってください。

ご指摘いただきありがとうございます。「元の配列」と「ソートされた添え字の配列」から「ソート済みの新しい配列」を生成する関数を作成します。

```
def sort_by_index(original_list, sorted_index_list):
    return [original_list[i] for i in sorted_index_list]
```

使用方法:

```
>>> original_list = [3, 1, 2, 5, 4]
>>> sorted_index_list = [1, 2, 0, 4, 3]
>>> sort_by_index(original_list, sorted_index_list)
[1, 2, 3, 5, 4]
```

この関数は、元の配列とソートされた添え字の配列を受け取り、ソート済みの新しい配列を生成します。
再度、ご指摘いただきありがとうございました。

YA

この実行結果は間違っています。上の例では、sorted_index_list[3] は 4、sorted_index_list[4] は 5 になるため、出力が [1, 2, 3, 4, 5] になります。修正してください。

申し訳ありません、実行例に誤りがありました。ご指摘いただきありがとうございます。

正しい例を提供します。

```
>>> original_list = [3, 1, 2, 5, 4]
>>> sorted_index_list = [1, 2, 0, 4, 3]
>>> sort_by_index(original_list, sorted_index_list)
[1, 2, 3, 4, 5]
```

このように、sorted_index_list の要素に対応する original_list の要素を選んで順に並べたものがソート済みの配列になります。
再度、ご指摘いただきありがとうございました。

まとめ

現状の ChatGPT はここまでできる

- **プログラミング言語の振る舞いをだいたい理解している**
  - 仕様の変更や追加の要件をどのような編集に落とし込むかをおおよそ理解している（凄い
  - 細かくてややこしい変更になってくると怪しい（初心者っぽい）
- **さっきの例でやってないけどできること**
  - ネット上にサンプルが多いコードやアルゴリズムはかなり高品質で生成できる
  - 別のプログラミング言語への翻訳はかなり上手くいく
- **こういう使い方ができそう**
  - アルゴリズムを要件に合わせて修正して初期コードを得て、そこから続きを書く。

2023年2月 に設立 20 周年になる名工大発 AI ベンチャー

自己紹介 & 会社紹介

山口陽平 何でも学び何でもやる人

有限会社来栖川電算　創業者・取締役・最高技術責任者

職務	全体	ビジョン・ビジネス・資金・設備・人員・制度・文化・パートナーシップの計画・遂行
	個別	企画，調査，研究，開発，運用，指導，教育，広報，採用，...
略歴	2001	名古屋工業大学 知能情報システム 卒業
	2003	名古屋工業大学 電気情報工学 修士課程 修了 有限会社来栖川電算 設立 IPA 未踏ソフトウェア創造事業 採択
	2012	名古屋工業大学 情報工学 博士課程 退学
	2013	Mashup Awards 9 優秀賞 受賞

仕事 = 趣味 何にでも面白みを見出す人

休日も論文読み・プログラミング・コミュニティ活動・講演・執筆

興味	ロボット，AI，UI，UX
専門	機械学習，信号処理，記号処理，プログラミング言語，コンパイラ，データベース，ソフトウェアアーキテクチャ，ソフトウェアプロセス
得意	分析，抽象化，高速化，省資源化，並列化，分散化

来栖川電算

設立 2003 年　名古屋工業大学発ベンチャー企業　従業員 115 人

- **様々な領域の顧客とともに彼らが抱える課題を解決**
  - AI・ML を応用した認識技術・自動化技術の研究開発
  - ソフトウェア・アノテーションの製造・販売
  - 研究開発を加速する基盤サービスの提供

情報検索

画像認識 API

商品パッケージで情報検索する API

情報検索

Cellars

ワインラベルで情報検索するアプリ

情報検索

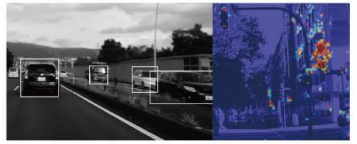
AR 付箋

書籍の中身を検索するアプリ

環境認識

走行データ解析

周辺環境を認識し、様々な解析に活用



環境認識

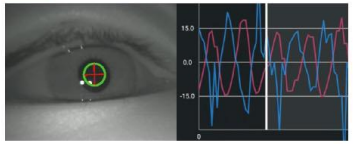
安全運転支援

高齢ドライバを対象に実証実験を実施

モーション認識

眼球運動推定1

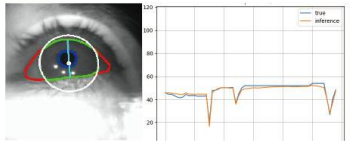
眼球運動を監視し、眠気の推定に活用



モーション認識

眼球運動推定2

眼球運動を監視し、眠気の推定に活用



文字認識

タンゴチュウ

写真に写る単語を抽出するサービス

環境認識

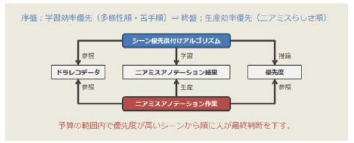
地図エディタ

地物・交通規則の整備を効率化

環境認識

ニアミス DB

商品企画と仕様策定に活用



モーション認識

生体情報推定

特別なセンサなしに生体情報を推定

モーション認識

動作推定 API

加速度センサで人の行動を推定

モーション認識

DZ Auth 顔追跡

隠れ・フレームアウトに対して頑健

環境認識

交通情報調査

道路沿いの視覚情報を収集し、活用

環境認識

生活情報調査

道路沿いの視覚情報を収集し、活用

モーション認識

なりきり 2.0

ヒーローの動きでゲームを制御

モーション認識

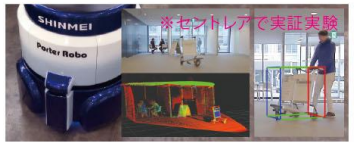
毎朝体操

腕の動きで体操採点するアプリ

環境認識

空港で働くロボット

カード回収・荷物運搬



環境認識

人流解析

人の動線改善・ロボットの経路計画

環境認識

広告マスキング

映像に写る広告をリアルタイムに除去

研究開発支援

GameControllerizer

プログラマブル・ゲームコントローラ

研究開発支援

DNN コンパイラ

DNN の推論を 10 ～ 1000 倍も効率化

研究開発支援

Ahab

大量の実験と計算資源を効率的に管理

研究開発支援

Annofab

3次元アノテーション対応！

高品質な教師データを大量に生産可能

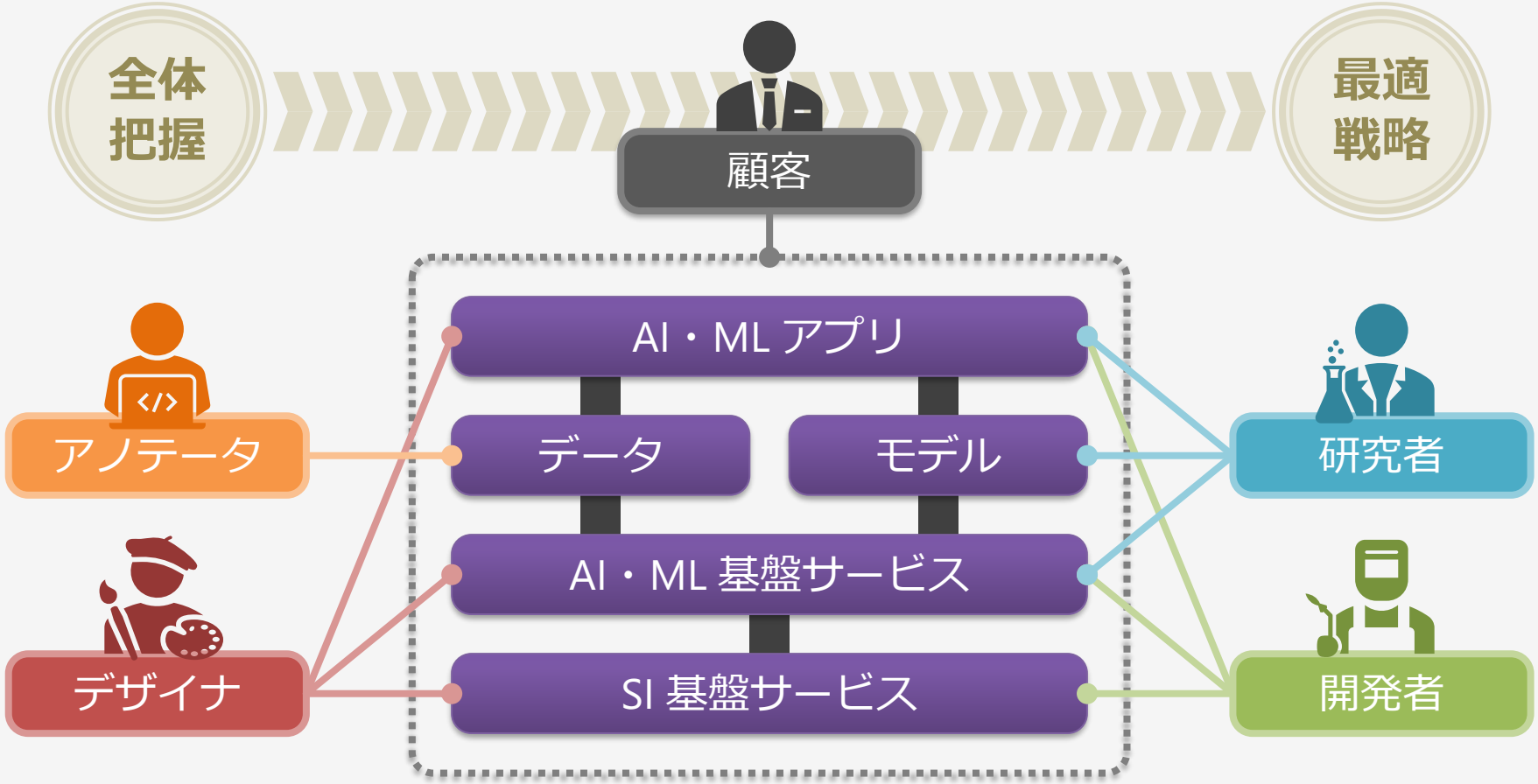
研究開発支援

X-nizer

特定情報マスキングサービス

顧客中心のワンストップ体制

AI・ML に関わる全工程を扱うため、適した工程で課題解決可能



ハイレベルな人材

未踏採択者，コンテスト受賞者，MVP 受賞者，OSS 貢献者，…

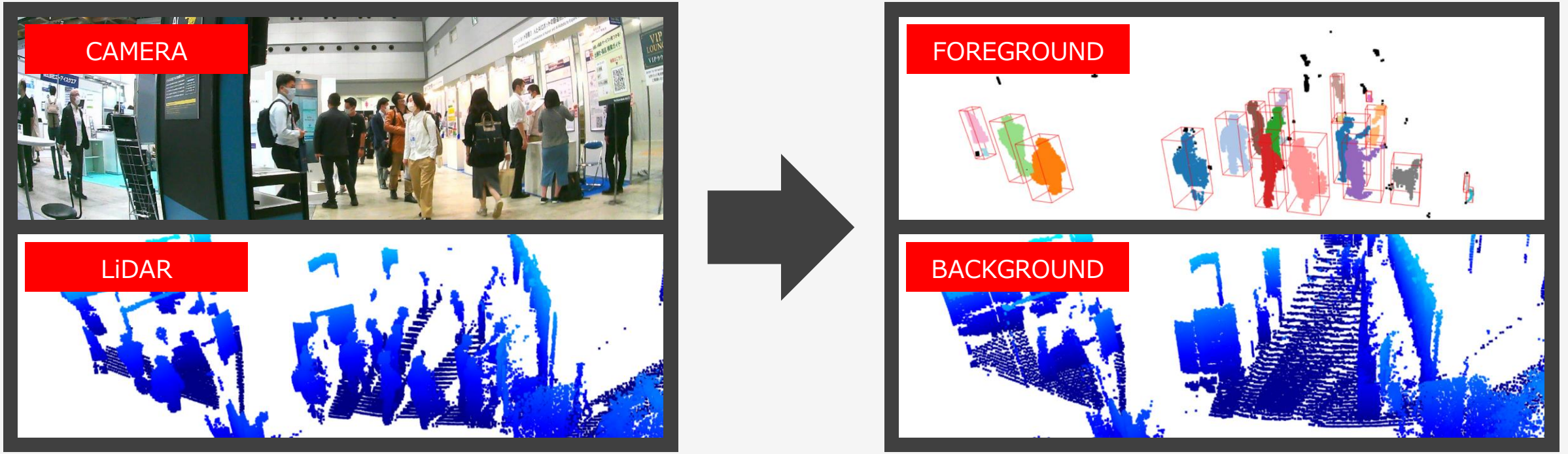
- **最先端の AI・ML 技術**
  - 文字認識，物体認識，行動認識，空間復元
- **限界性能を引き出す優れた実装技術**
  - 機械学習・コンパイラ・データベースなどのアルゴリズムやバイナリハックを駆使した高精度化・高速化・省資源化・並列化・分散化
- **高品質なソフトウェアを実現する技術**
  - アーキテクチャ・プロセスへの深い理解に基づく設計と計画
  - 徹底した自動化による品質と生産の改善

宣伝

研究開発インターンシップ

１～３名／年　研究開発チームにアシスタントとして参画し、実際の課題に取り組む

- **2022 年度のテーマ：３次元点群ストリームからの静止物と移動物の分離**



宣伝

機械学習 名古屋

https://machine-learning.connpass.com/

- **勉強会**　開催：隔月　対象：初心者（非技術者も含む）
  - アルゴリズムや事例の紹介・ハンズオン
- **研究会**　開催：毎月　対象：研究者，技術者
  - 機械学習関連の論文や記事の紹介，来栖川電算の研究者の他に日本各地の大学院生も参加

宣伝

コミュニティ活動

勉強会の主催，スタッフ・会場の提供，スポンサー・寄付，…

機械学習 名古屋

TFUG Tokyo

JAWS-UG

HEROES LEAGUE

OthloHack

NUPSC

JuliaTokai

PyCon JP

FP in Scala

Nagoya Frontend User Group

若手 Web 名古屋

名古屋合同懇親会

募集中

研究者

開発者

デザイナ

NGK2023S 連動企画

勉強会コミュニティに支えられて約 10 年
ラジオ体操採点アプリ「毎朝体操」から
今年も豪華景品をプレゼント！

「# 毎朝体操」付き Tweet 呟くだけ！
体操レポートを呟いた人を優遇！
賞品は他にもあるから要チェックだ ⇒

賞　品

特別賞

1 名様

ワシが育てた米

オリジナル Tシャツ
5 名様

オリジナル スプーン
5 名様

オリジナル ステンレス ボトル
5 名様

オリジナル タオル
5 名様