AutoSteer: Learned Query Optimization for Any SQL Database

Christoph Anneser\textsuperscript{1,2}, Nesime Tatbul\textsuperscript{2,3}, David Cohen\textsuperscript{2}, Zhenggang Xu\textsuperscript{4}
Prithviraj Pandian\textsuperscript{4}, Nikolay Laptev\textsuperscript{4}, and Ryan Marcus\textsuperscript{5}

\textsuperscript{1}TUM, \textsuperscript{2}Intel, \textsuperscript{3}MIT, \textsuperscript{4}Meta, \textsuperscript{5}University of Pennsylvania

VLDB, August 30, 2023
Many Database Management Systems expose **tunable optimizer knobs**.

- Usually belong to *rewrite rules* of the rule-based optimizer
- Can be used to steer query optimization

**SQL Query Hints**

- `SET enable_indexscan=false;` in /usr
- Hint-sets combine multiple hints. For example: `{indexscan:false, nestloop:false}`
Background – Steering Query Optimizers

Many Database Management Systems expose **tunable optimizer knobs**.

- Usually belong to **rewrite rules** of the rule-based optimizer
- Can be used to **steer** query optimization

---

```
SET enable_indexscan=false;
```
Background – Steering Query Optimizers

Many Database Management Systems expose **tunable optimizer knobs**.

- Usually belong to *rewrite rules* of the rule-based optimizer
- Can be used to *steer* query optimization

![Diagram showing the flow of Hints through SQL to the database](image-url)
Manually

Many Database Management Systems expose tunable optimizer knobs.

• Usually belong to rewrite rules of the rule-based optimizer
• Can be used to steer query optimization

Hints

SET enable_indexscan=false;

SQL Query
Many Database Management Systems expose **tunable optimizer knobs**.

- Usually belong to *rewrite rules* of the rule-based optimizer
- Can be used to *steer* query optimization

```
SET enable_indexscan=false;
```

**Hint-sets combine multiple hints.** For example: `{indexscan:false, nestloop:false}`
Background – Steering Query Optimizers

Many Database Management Systems expose **tunable optimizer knobs**.

- Usually belong to *rewrite rules* of the rule-based optimizer
- Can be used to *steer* query optimization

```sql
SET enable_indexscan=false;
```

**Hint-sets combine multiple hints.** For example: `{indexscan:false, nestloop:false}`
Many Database Management Systems expose **tunable optimizer knobs**.

- Usually belong to *rewrite rules* of the rule-based optimizer
- Can be used to *steer* query optimization

```
SET enable_indexscan=false;
```

**Hint-sets combine multiple hints.** For example: \{indexscan:false, nestloop:false\}
Lot of recent work on steered query optimizers:

- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al. [Industry]
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al. [Industry]
Lot of recent work on steered query optimizers:

- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al. [Industry]
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al. [Industry]
Background – Steering Query Optimizers Automatically

Lot of recent work on steered query optimizers:

- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al. [Industry]
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al. [Industry]
Lot of recent work on steered query optimizers:

- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al. [Industry]
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al. [Industry]
Lot of recent work on steered query optimizers:

- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al. [Industry]
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al. [Industry]
Lot of recent work on steered query optimizers:

- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al. [Industry]
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al. [Industry]
Lot of recent work on steered query optimizers:

- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al. [Industry]
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al. [Industry]
Limitations of Previous Approaches

- Databases usually expose up to hundreds of knobs
  - static, predefined or random hint-sets
Limitations of Previous Approaches

✗ Databases usually expose up to hundreds of knobs
   ⇒ static, predefined or random hint-sets

✗ Tight integration into the DBMS’ query optimizer
Limitations of Previous Approaches

✗ Databases usually expose up to hundreds of knobs
  ⇒ static, predefined or random hint-sets

✗ Tight integration into the DBMS’ query optimizer

✗ Requires good knowledge of the query optimizer
Limitations of Previous Approaches

✗ Databases usually expose up to hundreds of knobs
  ⇒ static, predefined or random hint-sets

✗ Tight integration into the DBMS’ query optimizer

✗ Requires good knowledge of the query optimizer

✗ Custom for every database
Limitations of Previous Approaches

- Databases usually expose up to hundreds of knobs
  ⇒ static, predefined or random hint-sets

- Tight integration into the DBMS’ query optimizer

- Requires good knowledge of the query optimizer

- Custom for every database

⇒ **AutoSteer is a generic framework to steer query optimizers!**
Limitations of Previous Approaches

- Databases usually expose up to hundreds of knobs
  ⇒ static, predefined or random hint-sets
- Tight integration into the DBMS’ query optimizer
- Requires good knowledge of the query optimizer
- Custom for every database

⇒ **AutoSteer is a generic framework to steer query optimizers!**

(A) Any SQL Database
\ e.g. PostgreSQL, PrestoDB, SparkSQL

SQL Queries

anneser@in.tum.de

AutoSteer: Learned Query Optimization for Any SQL Database
Limitations of Previous Approaches

❌ Databases usually expose up to hundreds of knobs
⇒ static, predefined or random hint-sets

❌ Tight integration into the DBMS’ query optimizer

❌ Requires good knowledge of the query optimizer

❌ Custom for every database

⇒ **AutoSteer is a generic framework to steer query optimizers!**
Limitations of Previous Approaches

- Databases usually expose up to hundreds of knobs
  ⇒ static, predefined or random hint-sets
- Tight integration into the DBMS’ query optimizer
- Requires good knowledge of the query optimizer
- Custom for every database

⇒ **AutoSteer is a generic framework to steer query optimizers!**
AutoSteer at a Glance

AutoSteer generates training data by exploring and executing alternative plans. It steers queries at runtime and uses the TCNN to infer execution times.

Notation:
- Training mode
- Inference mode
- All modes
- Database-specific
AutoSteer – Overview

**AutoSteer**

**DBMS**

**Config**

1. Knob 1
2. Knob 2
3. Knob 3

**Parser**

**Logical Optimizer**

**Physical Optimizer**

**Execution Engine**

**Notation**

- Database-specific

**AutoSteer** generates training data by exploring and executing alternative plans.

**Inference Mode**

AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers” → Demo Group B

**QO-INSIGHT**

- Upload
- Postgres
- TPC-H Q14
AutoSteer – Overview

AutoSteer

DBMS

Notation

- Database-specific

Knobs

Knob 1 = / Parser
Knob 2 = /reve Logical Optimizer
Knob 3 = /

Logical Optimizer

Physical Optimizer

Execution Engine

IR

exec

QO-INSIGHT

Upload

Postgres

TPC-H Q14

Greedy

HS Search

TCNN

SQL Queries

Hint-sets & Evaluation

Training Mode

AutoSteer generates training data by exploring and executing alternative plans.

Inference Mode

AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

VLDB 2023: "QO-Insight: Inspecting Steered Query Optimizers"
AutoSteer – Overview

**AutoSteer**

**Knobs**

- Knob 1:
- Knob 2:
- Knob 3:

**Parser**

**Logical Optimizer**

**Physical Optimizer**

**Execution Engine**

**Notation**

- Database-specific

**Connector**

**DBMS**

**Query Span**

A query span contains all the knobs that contribute to the query plan's optimization!

**Training Mode**

AutoSteer generates training data by exploring and executing alternative plans.

**Inference Mode**

AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

---

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers”

→ Demo Group B
AutoSteer - Generic DBMS Connector

**Required Functionality:**

- Set up database connection
- Toggle optimizer knobs
- Explain and analyze SQL queries
- Execute SQL queries
- Track execution times
AutoSteer - Generic DBMS Connector

Required Functionality:

- Set up database connection
- Toggle optimizer knobs
- Explain and analyze SQL queries
- Execute SQL queries
- Track execution times

class DBConnector:
    def set_knob(knob: str, enable: bool) -> None:
        """Disable the provided list of knobs""
        raise NotImplementedError()

    def get_knob(self, knob: str) -> bool:
        """Get current status of a knob""
        raise NotImplementedError()

    def explain(self, query: str) -> str:
        """Explain a query and return its plan""
        raise NotImplementedError()

    def execute(self, query: str) -> TimedResult:
        """Execute query and measure time""
        raise NotImplementedError()
AutoSteer - PostgreSQL Connector

```python
class PostgreSQLConnector(DBConnector):
    def __init__(url: str):
        self.conn = ... # setup PostgreSQL connection

    def set_knob(knob: str, enable: bool) -> None:
        self.conn.execute(f"SET {knob} TO {'ON' if enable else 'OFF'}")

    def execute(query: str) -> dict:
        return self.conn.execute(query)

    def explain(query: str) -> dict:
        return self.execute(f"EXPLAIN (FORMAT JSON) {query}")
```

anneser@in.tum.de AutoSteer: Learned Query Optimization for Any SQL Database 6
AutoSteer - PostgreSQL Connector

class PostgreSQLConnector(DBConnector):
    def __init__(url: str):
        self.conn = ... # setup PostgreSQL connection

    def set_knob(knob: str, enable: bool) -> None:
        self.conn.exec(f"SET {knob} TO {'ON' if enable else 'OFF'}")

    def execute(query: str) -> dict:
        return self.conn.exec(query)

    def explain(query: str) -> dict:
        return self.executeQuery(f'EXPLAIN (FORMAT JSON) {query}')

⇒ DB Connectors mainly differ in the syntax to toggle knobs and to explain plans!
class PostgreSQLConnector(DBConnector):
    def __init__(url: str):
        self.conn = ... # setup PostgreSQL connection

    def set_knob(knob: str, enable: bool) -> None:
        self.conn.exec(f"SET {knob} TO {'ON' if enable else 'OFF'}")

    def execute(query: str) -> dict:
        return self.conn.exec(query)

    def explain(query: str) -> dict:
        return self.execute(f"EXPLAIN (FORMAT JSON) {query}")

AutoSteer also supports integrated database connectors (AutoSteer-C).
AutoSteer – Overview

SQL Queries

AutoSteer

knobs.txt

Connector

DBMS

Config
Knob 1
Knob 2
Knob 3

Parser
IR
Logical Optimizer
IR
Physical Optimizer
exec
Execution Engine

Notation

Training mode
AutoSteer generates training data by exploring and executing alternative plans.

Inference mode
AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers” → Demo Group B

anneser@in.tum.de
AutoSteer: Learned Query Optimization for Any SQL Database
AutoSteer – Overview

AutoSteer

SQL Queries

Query Span

Connector

DBMS

Config

Knob 1=

Knob 2=

Knob 3=

Parser

Logical Optimizer

Physical Optimizer

Execution Engine

IR

IR

exec

Notation

Database-specific

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers”
→ Demo Group B

anneser@in.tum.de

AutoSteer: Learned Query Optimization for Any SQL Database
Query Span

A query span contains all the knobs that contribute to the query plan’s optimization!
AutoSteer – Overview

A query span contains all the knobs that contribute to the query plan’s optimization!

DBMS

SQL Queries

AutoSteer

Query Span

Knobs

Parser

Logical
Optimizer

Physical
Optimizer

Execution
Engine

Connector

Toggle a knob

DBMS

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers” → Demo Group B

anneser@in.tum.de
AutoSteer – Overview

A query span contains all the knobs that contribute to the query plan’s optimization!

DBMS

SQL Queries

Config

Parser

Logical Optimizer

Physical Optimizer

Execution Engine

Connector

Parser

Logical Optimizer

Physical Optimizer

Execution Engine

Query Span

Query Span

Toggle a knob

Knob 1=

Knob 2=

Knob 3=

Knobs .txt

Notation

- All modes
- Database-specific

_training mode_

AutoSteer generates training data by exploring and executing alternative plans.

_inference mode_

AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers”

anneser@in.tum.de

AutoSteer: Learned Query Optimization for Any SQL Database
AutoSteer – Overview

A query span contains all the knobs that contribute to the query plan’s optimization!
AutoSteer – Overview

SQL Queries

Greedy HS Search

Parser

Logical Optimizer

Physical Optimizer

Execution Engine

DBMS

Query Span

Connector

Notation

- All modes
- Database-specific

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers”

→ Demo Group B

anneser@in.tum.de

AutoSteer: Learned Query Optimization for Any SQL Database
AutoSteer – Overview

SQL Queries

AutoSteer

1. Query Span
2. Knob 1
3. Knob 2
4. Knob 3
5. Greedy HS Search

Connector

Toggle a knob

Configuration

DBMS

Parser
Logical Optimizer
Physical Optimizer
Execution Engine

Notation

- All modes
- Database-specific

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers”

QO-INSIGHT

Upload

Postgres

TPC-H

Q14

Greedy HS Search

TCNN

IR

IR

exec

Preprocess

Notation

Training mode

AutoSteer generates training data by exploring and executing alternative plans.

Inference mode

AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

AutoSteer: Learned Query Optimization for Any SQL Database
AutoSteer – Overview

SQL Queries → Query Span → Greedy HS Search

1. SQL Queries
2. Query Span
3. Greedy HS Search
4. Explained Query Plan
5. IR
6. Physical Optimizer
7. Execution Engine

DBMS

Config
Knob 1 = /
Knob 2 = reve
Knob 3 = /

Parser
Logical Optimizer
Physical Optimizer
Execution Engine

Notation
- All modes
- Database-specific

VLDB 2023: "QO-Insight: Inspecting Steered Query Optimizers" → Demo Group B

anneser@in.tum.de
AutoSteer: Learned Query Optimization for Any SQL Database
AutoSteer – Overview

SQL Queries

1. Query Span

2. Connector

3. Config

4. Logical Optimizer

5. Greedy HS Search

6. Physical Optimizer

7. Execution Engine

Notation:
- Training mode
- All modes
- Database-specific

DBMS

Toggle a knob

Knobs

Knob 1 = /reve

Knob 2 = /

Knob 3 = /

Connectors

0. knobs.txt

1. AutoSteer

2. Query Span

3. Logical Optimizer

4. Physical Optimizer

5. Greedy HS Search

6. Execution Engine

7. Result & Execution Time

A query span contains all the knobs that contribute to the query plan’s optimization!

Training Mode

AutoSteer generates training data by exploring and executing alternative plans.

Inference Mode

AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers”
Greedy Hint-Set Search

- There are $2^{\text{Query Span}}$ different hint-sets; However, most yield bad query plans.
Greedy Hint-Set Search

- There are $2^{|\text{Query Span}|}$ different hint-sets; however, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.

Input: SQL Query and Query Span (example: [$k_1$, $k_3$, $k_6$, $k_9$])
Greedy Hint-Set Search

- There are $2^{|\text{Query Span}|}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - **Assumption**: Larger, beneficial HSs consist of smaller, beneficial HSs
    
    *Not always, but in many cases true* – experimentally tested ✓
Greedy Hint-Set Search

- There are $2^{|\text{Query Span}|}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - **Assumption**: Larger, beneficial HSs consist of smaller, beneficial HSs
    - *Not always, but in many cases true* – experimentally tested ✓
- **Input**: SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)
Greedy Hint-Set Search

- There are $2^{|\text{Query Span}|}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - Assumption: Larger, beneficial HSs consist of smaller, beneficial HSs
    Not always, but in many cases true – experimentally tested ✔
- Input: SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)

Default Plan: $\emptyset \rightarrow 60s$
Greedy Hint-Set Search

• There are $2^{\text{Query Span}}$ different hint-sets; However, most yield bad query plans.
• Greedy search generates promising hint-sets (HS) with reasonable overhead.
  – **Assumption**: Larger, beneficial HSs consist of smaller, beneficial HSs
    
    *Not always, but in many cases true – experimentally tested ✔*

• **Input**: SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)

  Default Plan: \(\emptyset \rightarrow 60\text{s}\)

  \(\{k_1\} \rightarrow 45\text{s}\)
Greedy Hint-Set Search

- There are $2^{\text{Query Span}}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - **Assumption**: Larger, beneficial HSs consist of smaller, beneficial HSs
    
    *Not always, but in many cases true* – experimentally tested ✓
- **Input**: SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)

Default Plan: $\emptyset \rightarrow 60s$

- $\{k_1\} \rightarrow 45s$
- $\{k_3\} \rightarrow 65s$
- $\{k_1, k_3\} \rightarrow 71s$
- $\{k_6\} \rightarrow 20s$
- $\{k_1, k_9\} \rightarrow 33s$
- $\{k_3\} \rightarrow 65s$
- $\{k_1, k_6\} \rightarrow 80s$
- $\{k_6, k_9\} \rightarrow 120s$
- $\{k_1, k_9\} \rightarrow 71s$
- $\{k_6, k_9\} \rightarrow 120s$
Greedy Hint-Set Search

• There are $2^{|\text{Query Span}|}$ different hint-sets; However, most yield bad query plans.
• Greedy search generates promising hint-sets (HS) with reasonable overhead.
  – **Assumption**: Larger, beneficial HSs consist of smaller, beneficial HSs
    *Not always, but in many cases true* – **experimentally tested**
• **Input**: SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)

Default Plan: $\emptyset \rightarrow 60s$

- $\{k_1\} \rightarrow 45s$
- $\{k_3\} \rightarrow 65s$
- $\{k_6\} \rightarrow 20s$

$\{k_9\} \rightarrow 33s$
$\{k_3\} \rightarrow 65s$
$\{k_1, k_9\} \rightarrow 71s$
$\{k_6\} \rightarrow 80s$
$\{k_6, k_9\} \rightarrow 120s$
$\{k_1, k_6\} \rightarrow 80s$
$\{k_1, k_9\} \rightarrow 71s$
Greedy Hint-Set Search

- There are $2^{|\text{Query Span}|}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - **Assumption**: Larger, beneficial HSs consist of smaller, beneficial HSs
    
    *Not always, but in many cases true – experimentally tested ✓*

- **Input**: SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)

---

**Example: Default Plan**

\[ \emptyset \rightarrow 60s \]

- \(\{k_1\} \rightarrow 45s\)
- \(\{k_3\} \rightarrow 65s\)
- \(\{k_6\} \rightarrow 20s\)
- \(\{k_9\} \rightarrow 33s\)
Greedy Hint-Set Search

- There are $2^{|\text{Query Span}|}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - **Assumption**: Larger, beneficial HSs consist of smaller, beneficial HSs
    - *Not always, but in many cases true* – experimentally tested
- **Input**: SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)

---

Default Plan: \(\emptyset \rightarrow 60\)s

\[
\begin{align*}
\{k_1\} &\rightarrow 45\text{s} \\
\{k_3\} &\rightarrow 65\text{s} \\
\{k_6\} &\rightarrow 20\text{s} \\
\{k_9\} &\rightarrow 33\text{s}
\end{align*}
\]
Greedy Hint-Set Search

- There are $2^{|Query \ Span|}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - **Assumption**: Larger, beneficial HSs consist of smaller, beneficial HSs
    - *Not always, but in many cases true* – experimentally tested
- **Input**: SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)

```
Default Plan: $\emptyset \rightarrow 60s$

\{k_1\} \rightarrow 45s
\{k_3\} \rightarrow 65s
\{k_6\} \rightarrow 20s
\{k_9\} \rightarrow 33s

\{k_1, k_6\} \rightarrow 80s
```
Greedy Hint-Set Search

- There are $2^{|\text{Query Span}|}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - **Assumption:** Larger, beneficial HSs consist of smaller, beneficial HSs
    - *Not always, but in many cases true* – experimentally tested
- **Input:** SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)

![Diagram](image-url)
Greedy Hint-Set Search

- There are $2^{|\text{Query Span}|}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - Assumption: Larger, beneficial HSs consist of smaller, beneficial HSs
    - Not always, but in many cases true – experimentally tested
- Input: SQL Query and Query Span (example: \([k_1, k_3, k_6, k_9]\))
Greedy Hint-Set Search

- There are $2^{|\text{Query Span}|}$ different hint-sets; However, most yield bad query plans.
- Greedy search generates promising hint-sets (HS) with reasonable overhead.
  - Assumption: Larger, beneficial HSs consist of smaller, beneficial HSs
    - *Not always, but in many cases true* – experimentally tested
- **Input:** SQL Query and Query Span (example: $[k_1, k_3, k_6, k_9]$)

```
Default Plan: $\emptyset \rightarrow 60s$

$\{k_1\} \rightarrow 45s$  $\{k_3\} \rightarrow 65s$  $\{k_6\} \rightarrow 20s$  $\{k_9\} \rightarrow 33s$

$\{k_1, k_6\} \rightarrow 80s$  $\{k_1, k_9\} \rightarrow 71s$  $\{k_6, k_9\} \rightarrow 120s$
```
AutoSteer – Overview

A query span contains all the knobs that contribute to the query plan's optimization!

Training Mode
AutoSteer generates training data by exploring and executing alternative plans.

Inference Mode
AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

Notation
- Training mode
- All modes
- Database-specific

DBMS

Config

Knob 1
Knob 2
Knob 3

Parser
Logical Optimizer
Physical Optimizer
Execution Engine

IR
IR
exec

Greedy HS Search

Explained Query Plan

Result & Execution Time

AutoSteer

SQL Queries

Query Span

Connect

Toggle a knob

knobs .txt
AutoSteer – Overview

SQL Queries

Greedy HS Search

TCNN

Connector

Config

Knob 1=

Knob 2=

Knob 3=

DBMS

Parser

Logical Optimizer

Physical Optimizer

Execution Engine

Notation

- Training mode
- All modes
- Database-specific

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers”

→ Demo Group B
AutoSteer – Overview

AutoSteer generates training data by exploring and executing alternative plans.

---

Training Mode

AutoSteer generates training data by exploring and executing alternative plans.

Notation
- Training mode
- All modes
- Database-specific
**AutoSteer – Overview**

**Training Mode**

AutoSteer **generates training data** by exploring and **executing** alternative plans.

**Inference Mode**

AutoSteer **steers queries at runtime** and uses the TCNN to **infer execution times**.
AutoSteer – Overview

A query span contains all the knobs that contribute to the query plan's optimization!

Training Mode
AutoSteer generates training data by exploring and executing alternative plans.

Inference Mode
AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

**Notation**
- **Training mode**
- **Inference mode**
- **All modes**
- **Database-specific**
AutoSteer – Overview

1. SQL Queries
2. Query Span
3. Greedy HS Search
4. Explained Query Plan
5. Preprocess
6. IR
7. Result & Execution Time
8. Logical Optimizer
9. Physical Optimizer
10. TCNN

AutoSteer: Learned Query Optimization for Any SQL Database

VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers” → Demo Group B

 VLDB 2023: “QO-Insight: Inspecting Steered Query Optimizers” → Demo Group B
Evaluation
Evaluation of AutoSteer

- Tested AutoSteer with PostgreSQL, DuckDB, PrestoDB, MySQL, and SparkSQL

- Evaluation based on public benchmarks and production workloads at Meta
  - Join Order Benchmark (137 queries)
  - TPC-DS (100 queries)
  - Stackexchange (100 queries)
  - Dashboard application at Meta (>3000 queries, scanning PBs of data)

- Comparison to previous steering approaches (cf. to the paper for more details!)
Join Order Benchmark – PrestoDB

- For 137 queries, AutoSteer-C (w/ integrated connector) explored 1730 different hint-sets
- Evaluated between 8 and 34 different plans per query
- Achieved improvements of **up to 40%**

Relative performance changes of the best known alternative plan compared to the default plan.
Join Order Benchmark – PrestoDB

- AutoSteer-C (w/ integrated connector) using a tree convolutional neural network to infer execution times
- Reduces execution times of
  - unseen queries by 20.6% (opaque)
  - seen queries by 26.8% (transparent)

Relative performance changes of the optimal and the selected plan wrt. the default plan.
Dashboard Application at Meta – PrestoDB

- Focus on **tail latencies**
- >3000 Queries, scanning PBs of data, hundreds of worker nodes
- Workload runs multiple times per day

AutoSteer significantly reduces tail latencies of production workloads at Meta

![Graph showing comparison between Best Known Plan, AutoSteer’s Inference Mode, and PrestoDB across different percentiles and wall times.](image_url)
Greedy Exploration

- **Assumption**: Larger, beneficial HSs consist of smaller, beneficial HSs
- Greedy explores *significantly fewer hint-sets* than previous approaches (like Bao),
- but it meets or *outperforms* their performance improvements

![Diagram showing comparison between greedy exploitation and AutoSteer](attachment:diagram.png)
Conclusions

• AutoSteer is a practical framework to
  – steer query optimizers at runtime and to
  – find counter examples to debug and further improve existing query optimizers
Conclusions

• AutoSteer is a practical framework to
  – steer query optimizers at runtime and to
  – find counter examples to debug and further improve existing query optimizers

• AutoSteer outperforms previous steering approaches wrt.
  – exploration time, achieved performance improvements, and applicability to new systems
Conclusions

• AutoSteer is a practical framework to
  – steer query optimizers at runtime and to
  – find counter examples to debug and further improve existing query optimizers

• AutoSteer outperforms previous steering approaches wrt.
  – exploration time, achieved performance improvements, and applicability to new systems

• AutoSteer is open-source and already supports five DBMSs (https://github.com/IntelLabs/Auto-Steer)
Conclusions

- AutoSteer is a practical framework to
  - steer query optimizers at runtime and to
  - find counter examples to debug and further improve existing query optimizers

- AutoSteer outperforms previous steering approaches wrt.
  - exploration time, achieved performance improvements, and applicability to new systems

- AutoSteer is open-source and already supports five DBMSs (https://github.com/IntelLabs/Auto-Steer)

- Applied AutoSteer in an industrial setting – lessons learned in the paper!
Conclusions

- AutoSteer is a practical framework to
  - steer query optimizers at runtime and to
  - find counter examples to debug and further improve existing query optimizers

- AutoSteer outperforms previous steering approaches wrt.
  - exploration time, achieved performance improvements, and applicability to new systems

- AutoSteer is open-source and already supports five DBMSs (https://github.com/IntelLabs/Auto-Steer)

- Applied AutoSteer in an industrial setting – lessons learned in the paper!

Thank you for your attention!