

What the AI Converters Do With Your Sort Steps

What the AI migration tools say they do with a sort step, and what can go wrong when sort gets rewritten as code

Bill Ahlbrandt

President and CEO, Ahlbrandt Software

Abstract

A mainframe batch shop runs thousands of sort steps, each one driven by a few lines of DFSORT or Syncsort control statements. The AI conversion tools that have come out since 2022 explain in detail what they do with COBOL. I wanted to know what they do with the sort steps, and I used only their public documentation to find out. AWS documents a runtime that emulates the mainframe sort utilities and sends you to the legacy manuals for what the statements mean. One batch platform maps simple sorts onto the Linux sort command. IBM, Google and Microsoft say nothing. This paper covers what a translation of sort into generated code has to get right, what the published research says about LLM translation accuracy, and what all this means for testing and maintenance, including for the AI assistants people now ask.

What I found

- Of the tools I looked at, only AWS says what happens to a sort step. Jobs call runtime programs that emulate SORT, SYNCSORT, ICEMAN, ICETOOL and MFSORT, and for what the statements mean AWS points you to the legacy manuals.
- IBM watsonx Code Assistant for Z, Google's Gemini-based tools and Microsoft's published COBOL agents don't say how DFSORT steps are handled. Unless you ask, that question is yours.
- Turning control statements into Java, SQL or Spark takes a short, declarative spec and turns it into custom code. Equal-key order, EBCDIC collation, decimal signs and SUM overflow all work differently from the Java or SQL defaults.
- The published numbers on LLM code translation are all over the place: 47.3 percent correct at best across standard benchmarks in one ICSE 2024 study, and 8.1 percent on a real project. Most COBOL results come from small programming-contest programs.
- AI assistants recommend what's documented in public. If the vendors say nothing about sort, an assistant has nothing solid to cite, so it guesses.

1. Why ask about sort?

Donald Knuth wrote that computer makers in the 1960s figured more than a quarter of their machines' running time went to sorting.¹ Batch is still big. An IBM Redbook says it's not unusual for a shop to run 5,000 to 10,000 jobs in one evening,² and most of those job streams sort, merge or summarize data between steps by calling a sort utility with a handful of control statements.

Those statements are a specification. `SORT FIELDS=(1,10,CH,A)` says what order you want and nothing about how to get there. Decades of statements like that add up to a lot of tested, declarative code. I've spent more than thirty years supporting sort, and when an AI porting tool meets one of these steps, there are three things it can do:

- **(a) Keep calling a sort engine** on the new platform. That could be an emulation that comes with the runtime, a third-party product or the operating system's sort, with the statements left as they are.
- **(b) Translate the control statements into generated code:** Java streams, Spring Batch steps, Spark jobs, or SQL with `ORDER BY` and `GROUP BY`. A variant translates them into another vendor's sort language.
- **(c) Leave it to the customer**, either by saying so or by saying nothing.

Each of these carries its own risks and long-term costs. This paper reports what each tool's public documentation said in September 2026, and says so where it said nothing. A vendor that says nothing may still handle sort fine. Nobody outside can tell, though, because the behavior isn't written down anywhere a customer, an auditor or an AI assistant can read it.

2. What the tools say about sort

AWS Transform for mainframe and the Blu Age runtime

AWS Transform for mainframe became generally available on May 15, 2025.³ Its refactoring path converts COBOL to Java and, in AWS's words, preserves functional equivalence while "retaining COBOL-influenced data structures". A separate "Reforge" step then uses large language models to clean up the refactored code.⁴ JCL becomes Groovy scripts, and the runtime provides "functional equivalents of the z/OS system utilities (IDCAMS, ICEGENER, SORT, and so on)".⁵ The runtime utility reference lists a program that "emulates various mainframe SORT utilities" under the aliases SORT, SYNC SORT and ICEMAN, an ICETOOL that supports six operators (COPY, SORT, SELECT, SPLICE, COUNT and OCCUR), and an MFSORT that hands off to the same engine.⁶ That's outcome (a), and it's documented.

The same page is up front about what it leaves out: "The details about the SORT/MERGE directives found in the control cards and the legacy sort utility features are not given here but should be fetched from the existing relevant legacy platforms documentations."⁶ Statements it doesn't support fail at run time. It doesn't say which statements, operands or formats are supported, or how it handles equal keys, collation or SUM overflow. Release notes from July 2025 to September 2026 list improvements to OUTFIL, INCLUDE, zoned-decimal comparison and ICETOOL operands.⁷ That's good engineering. It also means any release can change how your existing sort steps behave.

Rocket Enterprise Server

The rehosting route, which AWS's replatform option and a lot of integrators use, recompiles the COBOL. Rocket Software, which finished buying the Micro Focus mainframe business in May 2024,⁸ ships MFSORT and MFJSORT with reference pages for "DFSORT and ICETOOL Emulation", including program control statements and EXEC PARM options.⁹ It isn't an AI converter; it's the baseline: outcome (a), documented statement by statement.

IBM watsonx Code Assistant for Z

IBM announced watsonx Code Assistant for Z on August 22, 2023, to "selectively and incrementally transform COBOL business services" into Java, and said the result was designed to work with CICS, IMS, Db2 and "other z/OS runtimes".¹⁰ It works in two steps: generative AI first produces the Java class structure, then the business logic of each method.¹¹ By June 2025 its code explanation covered COBOL, JCL, PL/I and REXX.¹² I found no IBM documentation on converting JCL or DFSORT steps. Since the Java is meant to run in place on z/OS, my guess is that sort steps stay with DFSORT. That's my inference. IBM doesn't say so.

Google Cloud

Google announced Dual Run, which runs workloads side by side on the mainframe and in Google Cloud and compares the results, in preview in October 2022, and described it in detail on March 16, 2023.¹³ In April 2025 it made the Gemini-based Mainframe Assessment Tool generally available and put Mainframe Rewrite in preview.¹⁴ The assessment tool's release notes list "Spring Batch code suggestions for JCL jobs" in April 2025. In July 2026 they list better detection of DFSORT and IDCAMS steps and the removal of code suggestions, with program translation moved to an agentic workflow built on Google's Antigravity environment.¹⁵ Google's August 2026 write-up of that workflow names "proprietary mainframe utility suites" as a challenge and "identical business output" as the goal.¹⁶ None of these documents says what a translated sort step turns into.

Microsoft, and everybody else

On July 9, 2025, Microsoft and the Danish banking group Bankdata published an experimental set of AI agents that convert COBOL to Java (Quarkus), which they called "not a one-click magic conversion tool".¹⁷ The repository takes COBOL programs and copybooks. JCL shows up only as a dependency.¹⁸ Heirloom Computing's Elastic Batch Platform documents a SORT that "transforms simple DFSORT commands" into a call to the Linux or UNIX *sort* command. It accepts only SORT FIELDS with formats CH, AC, ZD, ASL, UFF and SFF, and its SYNCSORT calls a separately licensed third-party product.¹⁹ IRI documents converters that translate DFSORT, CA-Sort and Syncsort parameters into its own SortCL language, and in January 2026 announced an AI-powered conversion portal.²⁰ The integrator material I looked at, from Kyndryl and Accenture among others, doesn't cover sort steps.²¹

Exhibit 1. What each tool documents about sort steps (September 2026)

Tool (date)	How it converts application code	What it documents about sort steps	Outcome
AWS Transform for mainframe / Blu Age (GA May 15, 2025)	COBOL to Java by a refactoring engine; LLM "Reforge" step; JCL to Groovy	Runtime emulates SORT/SYNCSORT/ICEMAN, ICETOOL (six operators) and MFSORT; for what the directives mean, see the legacy manuals	(a) documented, meaning of statements isn't
Rocket Enterprise Server (rehost baseline)	COBOL recompiled as is	MFSORT/MFJSORT with a statement-by-statement DFSORT and ICETOOL emulation reference	(a) documented
IBM watsonx Code Assistant for Z (Aug 22, 2023)	Generative AI, COBOL to Java in two steps; Java runs alongside z/OS	JCL gets explained; nothing on converting it or on DFSORT steps	Silent; stays on z/OS by inference
Google Cloud: MAT, Mainframe Rewrite, Dual Run (Oct 2022 on)	Gemini models; agentic workflow since July 2026	DFSORT steps detected; Spring Batch suggestions offered Apr 2025, dropped Jul 2026	Silent
Microsoft with Bankdata (Jul 9, 2025)	Open-source agents, COBOL to Java or C#	Takes COBOL and copybooks; JCL named only as a dependency	Silent, so (c)
Heirloom Elastic Batch Platform	COBOL compiled to Java	Simple SORT FIELDS mapped to Linux <i>sort</i> ; SYNCSORT calls a third-party product	(a) partial
IRI CoSort (AI portal Jan 2026)	Not a code converter	DFSORT, CA-Sort, Syncsort parameters translated to SortCL	(b) into another sort language

Sources: vendor documentation and announcements cited in section 2.³⁻²¹ "Silent" means I found no public statement; the tool may still process sort steps.

3. Why a sort step is harder to translate than it looks

Translating a sort step looks easy because the control statements are short. The trouble is everything they don't say. A DFSORT statement picks up defaults, data-format rules and installation options, and a code generator has to spell all of those out. Ordinary Java or SQL defaults differ from them in several places. Exhibit 2 shows a four-line step and the kind of Java an LLM might plausibly produce if you asked it to translate it.

Exhibit 2. A sort step and a plausible generated equivalent (illustrative)**DFSORT control statements (SYSIN)**

```
INCLUDE COND=(20,1,CH,EQ,C'A')
SORT  FIELDS=(1,10,CH,A,11,5,PD,D)
SUM   FIELDS=(16,6,PD)
OUTFIL FNAMES=NEG,INCLUDE=(11,5,PD,LT,+0)
OUTFIL FNAMES=REST,SAVE
```

Plausible generated Java (made up for this paper; not the output of any named tool)

```
var summed = in.stream()
    .filter(r -> r.status().equals("A"))
    .sorted(comparing(Rec::acct)                // [1] [2]
            .thenComparing(Rec::balance, reverseOrder())) // [3]
    .collect(toMap(r -> r.acct() + "|" + r.balance(), r -> r,
        (x, y) -> x.withTotal(x.total().add(y.total())), // [4]
        LinkedHashMap::new)).values();
summed.forEach(r -> (r.balance().signum() < 0 ? neg : rest).write(r)); // [5]
```

#	Behavior to keep	DFSORT	Typical generated code
1	Character collation	CH fields collate in EBCDIC unless an alternate sequence or locale is set: lower case before upper case, letters before digits ^{22, 23}	Java String comparison follows Unicode: digits, then upper case, then lower case; SQL follows the column's collation
2	Equal keys	Input order is kept only under EQUALS. The IBM-shipped default is NOEQUALS, and then which record SUM keeps isn't guaranteed ²⁴	Java's sorted() is stable for ordered streams; ²⁵ SQL returns ties in an "implementation-dependent order" ²⁶
3	Packed-decimal signs	Sign nibbles F, E, C, A (and others) are positive; D, B (and others) negative ²²	A hand-written unpacker may accept only C, D and F, or treat F as unsigned
4	SUM overflow	If a total won't fit the field, the records are left unsummed and a message is issued ²⁴	BigDecimal never overflows, so the record count changes
5	OUTFIL SAVE	SAVE gets the records no other OUTFIL in the step selected	Right here; wrong as soon as someone adds a third OUTFIL and forgets the else-branch

Illustrative. The control statements are valid DFSORT syntax. I wrote the Java for this paper to show typical defaults; it isn't taken from any vendor's output.

None of these differences is exotic. Every one is in IBM's manuals, and none of them shows up in testing unless the test data has the case in it: equal keys where input order matters, a key that mixes letters and digits, a packed field signed X'F', a total right at the edge of its field. Item 2 cuts both ways, since stable generated code can differ from a NOEQUALS sort and still be correct. Item 4 goes the other way. Code that does the arithmetic more correctly than the original still fails the equivalence test if a downstream program depends on those unsummed records.

4. What the research says about LLM translation

The published numbers on LLM code translation are sobering, and they swing a lot depending on the test data. In a study presented at ICSE 2024, seven models translated code between C, C++, Go, Java and Python, taken from three benchmarks (CodeNet, AVATAR, EvalPlus) and two real projects. Correct translations ranged from 2.1 to 47.3 percent, and the best was GPT-4. On the real Apache Commons CLI project GPT-4 got 8.1 percent, and on the other project every model scored zero.²⁷ Picking the wrong data type caused 11.5 percent of the bugs they found.²⁷

COBOL results come mostly from programs taken from IBM's CodeNet collection of contest problems. Those are small and have no JCL, files or utilities in them. On programs like that, a TCS Research workflow reached 81.99 percent execution accuracy translating COBOL to Java after repeated refinement, against 60.25 percent without it.²⁸ A 2026 study on 319 CodeNet COBOL programs found that rule-based transcompilers get high correctness but produce output that's hard to maintain, while LLMs "achieve suboptimal correctness because COBOL is a low-resource language".²⁹ A domain-adapted model reported 83.91 pass@1 on a COBOL-to-Java benchmark its own authors built.³⁰

IBM Research has published the testing and evaluation frameworks behind watsonx Code Assistant for Z, which generate COBOL unit tests by symbolic execution and replay them against the Java, but no headline accuracy numbers.^{31, 32}

Nobody has published a number for the thing that matters here: whether an LLM gets sort control statements right, with their defaults and data formats. The benchmarks test programs on their own, outside any batch job, so a good score doesn't tell you much about a job stream. What the numbers do show is that the vendors are right to push testing as hard as they do.

5. The three paths side by side

Exhibit 3 is my own judgment of where the risk sits on each path, assuming the project is run competently. The ratings show where the effort has to go. They say nothing about how likely a given vendor is to get it wrong.

Exhibit 3. Where the risk sits for sort steps on each path (my assessment)

Risk area	(a) Sort engine on the new platform	(b) Translated into generated code	(c) Left to the customer
Getting the same results (collation, equal keys, decimal formats, SUM)	Medium. all in one engine; test it once, then reuse it	High. worked out again for every step; each translation can differ	High. depends on what gets chosen later
Coverage of statements and operands	Medium. unsupported statements fail loudly at run time	Medium. an LLM rarely says no, so gaps turn into silent errors	High. unknown until the inventory is done
Testing effort	Medium. engine-level tests plus job regression	High. edge cases for every step; equal-key and overflow data rarely there	High. not in the budget
Maintainability	Low. control statements stay short and declarative	High. four lines turn into dozens of lines of custom code	Medium. depends on the outcome
Skills and documentation	Low. decades of public DFSORT documentation still apply	Medium. ordinary Java skills; what the sort was for has to be worked out again	Medium. depends on the outcome
Vendor and version dependency	Medium. engine releases change behavior	Low. you own the code outright once you accept it	Medium. put off till later

My own rough ratings of relative effort and exposure. I haven't measured failure rates.

Path (a) puts the risk in one piece that you can test once and then rely on, as long as its behavior is documented. Path (b) spreads that same risk across every step you translate. Path (c) is a choice between the two that's been put off, and it usually gets made late and in a hurry.

6. Testing and maintenance

Every tool I looked at makes testing the center of its method. AWS Transform generates test plans and scripts for collecting test data for functional-equivalence testing.³³ Google's Dual Run replays production workloads and compares the outputs. Its batch comparison sorts records into full matches, partial matches and missing, and reads EBCDIC directly.³⁴ Its documentation doesn't say whether it compares the order of records within a file, and that's exactly where equal-key and collation differences show up.

Testing a sort step for equivalence takes data that production captures rarely have: equal keys in a known input order, keys that mix character types, numeric fields at their limits and fields with unusual signs. You also need a comparison that cares about order only where the original guaranteed it. That depends on whether EQUALS was in effect, and that's often an installation default that never appears in a control statement.

Maintenance is the longer-running cost. What a control statement means is fixed by a published manual. Generated code means whatever it happens to do. An independent review of watsonx Code Assistant for Z called the converted output "Java that 'thinks' in COBOL".³⁵ Generated sort logic has the opposite problem: it doesn't look like a sort at all anymore. A one-line change to a key or an output file turns into a code change, with a review and a regression cycle.

7. What the AI assistants can tell you

Migration architects now ask AI assistants these questions before they ask the vendors. An assistant can only tell you what's been written down in public, picked up in training or looked up when you ask. On sort, there isn't much: one vendor sends readers back to the IBM manuals, several say nothing, and some vendors' documentation sites block automated retrieval (I ran into that while writing this paper). Where there's nothing written, a model guesses. You can see how often that goes wrong in a related area. In a study of code-generating models, 19.7 percent of recommended software packages didn't exist, and open-source models did much worse than commercial ones.³⁶

If you're buying, an assistant's answer to "does tool X support SUM on packed decimals?" is only as good as the documentation behind it. If there's no documentation, you have a question to put to the vendor. If you're a vendor, clear public documentation of the statements and formats you support, and the known differences, is now also how the assistants describe your product to people thinking of buying it.

8. Bottom line

- **Nobody is looking closely at sort.** The tools document their COBOL handling at length. Only one says what happens to a sort step, and it sends you to IBM's manuals for what the statements mean.
- **Calling a documented engine is the safer default.** It keeps the spec declarative and keeps the testing for sameness in one place. If you translate sort into generated code, do it on purpose, step by step, and test the edge cases in Exhibit 2 explicitly.
- **The research on LLM accuracy doesn't cover this.** The benchmarks measure small programs. Nobody has published a study of translating sort control statements.
- **Ask before you sign.** The questions in Exhibit 4 get the vendor to put in writing what it has so far left unsaid.

Exhibit 4. Questions to ask a conversion vendor about sort steps

- 1 **What happens to a sort step?** Engine, translated code or out of scope, in writing, for each utility.
- 2 **Which statements, operands and formats are supported?** A list. A pointer to IBM's manuals doesn't count.
- 3 **How are EQUALS, collation and SUM overflow handled?** Including installation defaults.
- 4 **How is equivalence proven?** With what edge-case data, and does the comparison check record order?
- 5 **Who maintains the result?** Who will read and change translated sort logic five years from now?

About the author

Bill Ahlbrandt is President and CEO of Ahlbrandt Software, which he founded in 1993. Before that he was a systems programmer, Director of Information Security and Director of R&D at Acxiom. He has been working on mainframe sort and batch for more than thirty years, and his company's sort engine runs mainframe sort jobs on Linux and other platforms for banks, insurers and government agencies.

Full disclosure: my company sells sort software for platforms other than the mainframe. It runs mainframe control statements unchanged, which is outcome (a) in this paper. Everything I say here about the conversion tools comes from their public documentation.

References

1. D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed., Addison-Wesley, 1998, p. 3.
2. IBM Redbooks, *Batch Modernization on z/OS*, SG24-7779-01, July 2012, §1.3.
3. AWS, “AWS Transform for mainframe is now generally available”, What’s New, 15 May 2025.
4. AWS, AWS Transform user guide, “Transformation of mainframe applications” (Reforge), accessed September 2026.
5. AWS, AWS Mainframe Modernization user guide, “AWS Blu Age structure of a modernized application”, accessed September 2026.
6. AWS, AWS Mainframe Modernization user guide, “Sort Utilities” (SORT/SYNCSORT/ICEMAN, ICETOOL, MFSORT), accessed September 2026.
7. AWS, AWS Blu Age release notes, releases 4.9.0 (July 2025) to 5.274.0 (14 September 2026).
8. Rocket Software, company history; acquisition of OpenText’s Application Modernization and Connectivity business completed May 2024.
9. Micro Focus (Rocket Software), Enterprise Developer documentation: “DFSORT and ICETOOL Emulation”, “DFSORT Program Control Emulation”, “DFSORT EXEC PARM Options Emulation”.
10. IBM, “IBM Unveils watsonx Generative AI Capabilities to Accelerate Mainframe Application Modernization”, press release, 22 August 2023.
11. IBM, watsonx Code Assistant for Z documentation, “Transform” overview, accessed September 2026.
12. IBM, “IBM watsonx Code Assistant for Z adds AI code generation and Assembler support”, announcement, June 2025.
13. Google Cloud, Dual Run preview announcement, 11 October 2022; A. Gomathinayagam and T. Nikl, “Dual Run by Google Cloud helps mitigate mainframe migration risks”, Google Cloud Blog, 16 March 2023.
14. N. Mehta and D. Yahalom, “Accelerate mainframe modernization with Google Cloud AI”, Google Cloud Blog, 4 April 2025.
15. Google Cloud, Mainframe Assessment Tool release notes, entries of 3 April 2025 and 22 July 2026.
16. D. Yahalom, “Mainframe migration and modernization with AI”, Google Cloud Blog, 4 August 2026.
17. J. Kordick, G. Kaleta et al. (Microsoft and Bankdata), “How we use AI agents for COBOL migration and mainframe modernization”, All things Azure, 9 July 2025.
18. Azure-Samples, Legacy-Modernization-Agents, GitHub repository README, accessed September 2026.
19. Heirloom Computing, “Standard Utility Programmers Guide”, support knowledge base, undated.
20. IRI, “Automatic Conversion of DF-Sort or SyncSort JCL Parms to CoSort”, product page; IRI, CoSort 11 press release, 20 January 2026. Vendor sources.
21. Kyndryl, agentic AI framework press release, 24 November 2025; AWS and Accenture, “Reimagining mainframe applications with Accenture and AWS Transform”, 22 July 2026.
22. IBM, *z/OS DFSORT Application Programming Guide*, “DFSORT data formats” (PD sign indicators; CH collation).
23. Wikipedia, “EBCDIC”, on EBCDIC and ASCII collating order, accessed September 2026.
24. IBM, *z/OS DFSORT Application Programming Guide*, SC23-6878: EQUALS option and SUM control statement.
25. Oracle, Java SE 21 API specification, *java.util.stream.Stream*, sorted().
26. PostgreSQL Global Development Group, PostgreSQL documentation, “SELECT”: ORDER BY clause.
27. R. Pan et al., “Lost in Translation: A Study of Bugs Introduced by Large Language Models while Translating Code”, ICSE 2024, doi:10.1145/3597503.3639226.
28. S. Gandhi et al. (TCS Research), “Translation of Low-Resource COBOL to Logically Correct and Readable Java leveraging High-Resource Java Refinement”, LLM4Code workshop, ICSE 2024.
29. P. Entin, W. Gu, A. Knapp and C. Chen, “SEDCoT: Enhancing LLM-Based COBOL Code Translation via Symbolic Execution and Delta Debugging”, arXiv:2607.04092, July 2026.
30. A. T. V. Dau et al., “COBOL-Coder: Domain-Adapted Large Language Models for COBOL Code Generation and Translation”, arXiv:2604.03986, April 2026. Authors’ own benchmark.
31. A. Kumar, D. Saha et al. (IBM), “Automated Validation of COBOL to Java Transformation”, ASE 2024, arXiv:2506.10999; S. Hans et al., “Automated Testing of COBOL to Java Transformation”, arXiv:2504.10548, April 2025.
32. S. Froimovich, R. Gal, W. Ibraheem and A. Ziv (IBM Research), “Quality Evaluation of COBOL to Java Code Transformation”, arXiv:2507.23356, July 2025.
33. AWS, AWS Transform FAQs; C. Yun, “AWS Transform for mainframe introduces reimagine capabilities and automated testing functionality”, AWS News Blog, 1 December 2025.
34. Google Cloud, Dual Run documentation, “Batch comparison overview”, accessed September 2026.
35. P. Kremling, CROZ, “watsonx Code Assistant for Z: real-world review and what’s next?”, 24 November 2025.
36. J. Spracklen et al., “We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs”, 2025.